GUÍA PRÁCTICA DE LOS INVEN-KIT BÁSICO, COMPLETO Y AVANZADO 2016

BY INVEN

CONTENIDO DEL KIT

DESGLOSE DEL INVEN KIT AVANZADO

inven

1. Compatible Arduino UNO R3 con cable
2. 1 x Cable USB para arduino.
3. 1x Protoboard 830puntos
4. 1 x Kit jumper wires.
5. Pack de Leds: 5 de cada color rojo, amarillo y verde.
6. Pack Resistencias: 10 de cada valor 220 Ω, 1 kΩ, 10 kΩ.
7. 5 x Pulsador con tapa amarilla.
8. 1 x Sensor de llama (Flame sensor).
9. 2 x Interruptor tilt SW-520D de bola.
10. 3 x Sensor de luz GL55 (Fotoresistencia).
11. 1 x Buzzer activo y 1x Buzzer pasivo.
12. 1 x Matriz de pulsadores 4 x 4
13. 1 x LM35DZ (Sensor de temperatura).
14. 1 x 74HC595 (Registro de desplazamiento 8 -bits).
15. 1 x Receptor IR con Mando a distancia.
16. 1 x Matriz de led (8x8).
17. 1 x Display led 1 dígito y 4 dígitos/ 7 segmentos.
18. 1 x Portapilas (Batería no incluida).
19. 1 x Módulo led RGB.
20. 1 x Potenciometro 10 kΩ.
21. 1 x Display LCD1602 con I2C.
22. 1 x Módulo RTC.
23. 1 x Servomotor sg-90.
24. 1 x Motor Paso a paso con Driver.
25. 1 x Módulo RFID con Tarjeta de plástico y llavero 125Khz.
26. 1 x Módulo Joystick.
27. 1 x Sensor de Sonido.
28. 1 x Sensor de agua.
29. 1 x Módulo dht-11.
30. 1 x Módulo Relé 1 canal.
31. 1 x Sensor de movimiento PIR.
32. 1 x Módulo ultrasonidos de distancia.
33. 1 x Módulo Giroscopio MPU-6050.
34. 2 x Módulo de comunicaciones nRF2401.
35. 1 x Módulo Ethernet ENC28J60
36. 1 x Módulo tarjeta microsd
37. 1 x Módulo bluetooth HC-05
38. 1 x Módulo GPS NEOMV2.

EXPLICACIÓN DE CADA MÓDULO

3. **Protoboard 830puntos**

La Protoboard o breadboard es uno de los primeros pasos a la electrónica, en el podrás realizar circuitos sin la necesidad de soldar, posee 630 puntos de conexión para componentes y 200 puntos para la alimentación.

4. **1 x Kit jumper wires.**

Conjunto de cables finos y flexibles macho o hembra para realizar con facilidad conexiones desde nuestra placa Arduino a una protoboard. Vienen en diversos colores para que puedan ser identificados fácilmente.

5. **Pack de Leds: 5 de cada color rojo, amarillo y verde.**

Un LED es un diodo emisor de luz, es decir, un tipo particular de diodo que emite luz al ser atravesado por una corriente eléctrica. Los diodos (emisor de luz, o no) son unos de los dispositivos electrónicos fundamentales.

6. **Pack Resistencias: 10 de cada valor 220 Ω, 1 kΩ, 10 kΩ.**

Se denomina resistencias o resistor al componente electrónico diseñado para introducir una resistencia eléctrica determinada entre dos puntos de un circuito eléctrico.

7. **5 x Pulsador con tapa amarilla.**

Los pulsadores son componentes eléctricos digitales pasivos que cortan o permiten el paso de la señal eléctrica mientras se encuentren pulsados.

8. **1 x Sensor de llama (Flame sensor).**

El sesor de llama se trata de un dispositivo analógico pasivo que permite detectar la existencia de llamas/fuego a través del espectro de luz que emite.

9. **2 x Interruptor tilt SW-520D de bola.**

El sensor de inclinación tilt se trata de un dispositivo de inclinación que proporciona una señal digital (0 ó 1) en caso de su inclinación supere un umbral

10. 3 x Sensor de luz GL55 (Fotoresistencia).

Un fotoresistor, o LDR (light-dependent resistor) es un dispositivo analógico pasivo cuya resistencia varia en función de la luz recibida.

11. 1 x Buzzer activo y 1x Buzzer pasivo.

Los buzzer, denominados también zumbadores, son dispositivos que generan un sonido cuando son conectados a tensión.

Un buzzer activo incorpora un oscilador simple por lo que únicamente es necesario suministrarle corriente para que emita sonido fijo. En oposición, los buzzer pasivos necesitan recibir una onda a la frecuencia deseada.

12. 1 x Matriz de pulsadores 4 x 4.

Un teclado matricial es un simple arreglo de botones conectados en filas y colúmnas, de modo que se pueden leer varios botones con el mínimo número de pines requeridos. Un teclado matricial 4x4 solamente ocupa 4 lineas de un puerto para las filas y otras 4 lineas para las colúmnas, de este modo se pueden leer 16 teclas utilizando solamente 8 líneas de un microcontrolador

13. 1 x LM35DZ (Sensor de temperatura).

El LM35DZ es un sensor de temperatura digital. Se trata de un integrado con su propio circuito de control, que proporciona una salida de voltaje proporcional a la temperatura.

14. 1 x 74HC595 (Registro de desplazamiento 8 -bits).

El 74HC595 es un Shift Register de 8 bits serial-in, parallel-out, pertenece a una familia de chips que aceptan una entrada de bits en serie y los sacan en 8 pines paralelos. Solo sirve para escribir señales digitales y no para leerlas.

15. 1 x Receptor IR con Mando a distancia.

El mando a distancia emisor IR gobierna un LED infrarrojo muy similar a los LEDs normales, solo que diseñados para emitir luz en un color que no vemos.

Los receptores de infrarrojos son unos pequeños integrados que poseen una foto-célula que detecta emisiones infrarrojas.

16. 1 x Matriz de led (8x8).

Una matriz LED es un display formado por múltiples LED en distribución rectangular. Existen distintos tamaños, siendo el más habitual los cuadrados de 8×8 LED.

17. 1 x Display led 1 dígito y 4 dígitos/ 7 segmentos.

Los display de 7 segmentos son componentes que se utilizan para representar caracteres (normalmente números) en muchos dispositivos electrónicos. Están constituidos por una serie de leds de tal forma que formen un número '8'.

18. 1 x Portapilas (Batería no incluida).

Portapila para pila de 9v con conector de 2.1mm para alimentar el arduino

19. 1 x Módulo led RGB.

Los led RGB estan compuestos por 3 leds en el mismo encapsulado. Estos leds están compuestos de leds de colores primarios: rojo (Red), verde (Green), y azul (Blue), al variar la intensidad de corriente de cada led se producen diferentes colores.

20. 1 x Potenciometro 10 kΩ.

Un potenciómetro es una resistencia o resistor que varía su valor por acción mecánica.

21. 1 x Display LCD1602 con I2C.

El LCD(Liquid Crystal Dysplay) o pantalla de cristal líquido es un dispositivo empleado para la visualización de contenidos o información de una forma gráfica, mediante caracteres, símbolos o pequeños dibujos dependiendo del modelo. Está gobernado por un microcontrolador el cual dirige todo su funcionamiento.

22. 1 x Módulo RTC.

Basado en el chip DS1307 de Maxim, este módulo de reloj de tiempo real lleva la cuenta de segundos, minutos, horas, día, mes y año automáticamente y válido hasta el 2100

23. 1 x Servomotor sg-90.

Un servo es un tipo motor al cual le indicamos directamente el ángulo deseado y el servo se encarga de posicionarle en este ángulo.

24. 1 x Motor Paso a paso con Driver.

El 28BYJ-48 es un pequeño motor paso a paso bipolar que incorpora una reductora integrada. El 28BYJ-48 tiene un paso de 5.625 grados (64 pasos por vuelta). Con una precisión total es de 4096 pasos por vuelta, equivalente a un paso de 0.088º, que es una precisión muy elevada.

25. 1 x Módulo RFID con Tarjeta de plástico y llavero 125Khz.

Este módulo esta basado en el chip MFRC522 para lectura y escritura de tags RFID de 125Khz en diferentes condiciones y con control de errores.

26. 1 x Módulo Joystick.

El módulo Joystick es un dispositivo que consta de 2 potenciometros que permiten conocer la posición exacta de la palanca. El Módulo Joystick posee también un "Eje Z" que no es otra cosa más que un botón que cierra el circuito al presionar la palanca, este se puede conectar a cualquier pin digital o analógico.

27. 1 x Sensor de Sonido.

Este módulo se trata de un sensor activado por sonido. Consta de un micrófono que recoge la señal de audio o ruido ambiente. Esta señal es amplificada pudiendo activar un pulso lógico al alcanzar cierto umbral o recogerse a como un valor alógico entre 0 y 1023.

28. 1 x Sensor de agua.

Un sensor de agua es un módulo que mide la presencia y cantidad de agua que lo envuelve. Muy utilizado para conocer la presencia de escapes de agua o para saber si hay condensación en una cierta zona.

29. 1 x Módulo dht-11.

Este sensor dispone de un procesador interno que realiza el proceso de medición, proporcionando la medición de temperatura y humedad mediante una señal digital, por lo que resulta muy sencillo obtener la medición desde un microprocesador como Arduino.

30. 1 x Módulo Relé 1 canal.

Un relé es un dispositivo electromecánico que permite a un procesador como Arduino controlar cargas a un nivel tensión o intensidad muy superior a las que su electrónica puede soportar. Con una salida por relé podemos encender o apagar cargas de corriente alterna a 220V e intensidades de 10A, lo cual cubre la mayoría de dispositivos domésticos que conectamos en casa a la red eléctrica.

31. 1 x Sensor de movimiento PIR.

Los sensores infrarrojos pasivos (PIR) son dispositivos para la detección de movimiento. Son baratos, pequeños, de baja potencia, y fáciles de usar. Por esta razón son frecuentemente usados en juguetes, aplicaciones domóticas o sistemas de seguridad.

32. 1 x Módulo ultrasonidos de distancia.

Un sensor de ultra sonidos es un dispositivo para medir distancias. Su funcionamiento se base en el envío de un pulso de alta frecuencia, no audible por el ser humano. Este pulso rebota en los objetos cercanos y es reflejado hacia el sensor, que dispone de un micrófono adecuado para esa frecuencia.

33. 1 x Módulo Giroscopio MPU-6050.

El circuito integrado MPU-6050 contiene un acelerómetro y giroscopio MEMS en un solo encapsulado. Cuenta con una resolución de 16-bits que se aplica para cada eje X, Y y Z al igual que en la velocidad angular. El sensor es ideal para diseñar control de robótica, medición de vibración, sistemas de medición inercial (IMU), detector de caídas, sensor de distancia y velocidad, y muchas cosas más. El MPU-6050 contiene un giroscópico, un acelerómetro, además de un sensor de temperatura. Mediante I2C regresa unos valores conocidos como raw o "crudos" según el registro seleccionado

34. 2 x Módulo de comunicaciones nRF2401.

El nRF24L01 es un circuito transceptor de RF de 2Mbps para la banda ISM de 2,4 GHz de ultra bajo consumo. El módulo soporta una interfaz SPI de alta velocidad para el controlador de la aplicación. No se requieren filtros externos, resonadores, o diodos VCO.

35. 1 x Módulo Ethernet ENC28J60.

El Módulo Ethernet ENC28J60 es popular por su capacidad para conectar y controlar nuestro Arduino a la red local o internet. A pesar de consumir gran cantidad de memoria, es muy útil para aplicaciones básicas.

36. 1 x Módulo tarjeta microsd.

Este módulo nos permite leer y escribir datos en una tarjeta de memoria tipo Micro SD que son las más comunes en el mercado. El modulo se puede alimentar con 3.3V o 5V usando los pines respectivos

37. 1 x Módulo bluetooth HC-05.

El módulo Bluetooth HC-05 es popular por su capacidad para trabajar tanto como esclavo (los dispositivos se conectan a él) como maestro (iniciando la comunicación). Ampliamente utilizado para aplicaciones en domótica, robótica y educación pudiendo controlar el arduino desde el móvil a través de aplicaciones publicadas por terceros o creadas por nosotros con herramientas tipo appinventor.

38. 1 x Módulo GPS NEOMV2.

El módulo GPS en su modelo GY-GPS6MV2 viene con un módulo de serie U-Blox NEO 6M equipado en la PCB, una EEPROM con configuración de fábrica, una pila de botón para mantener los datos de configuración en la memoria EEPROM, un indicador LED y una antena cerámica. También posee los pines o conectores Vcc, Rx, Tx y Gnd por el que se puede conectar a algún microcontrolador mediante una interfaz serial. Para que nuestro módulo GPS funcione a la perfección se recomienda hacer las pruebas en un ambiente abierto o cercano a la ventana para una correcta recepción de la señal.

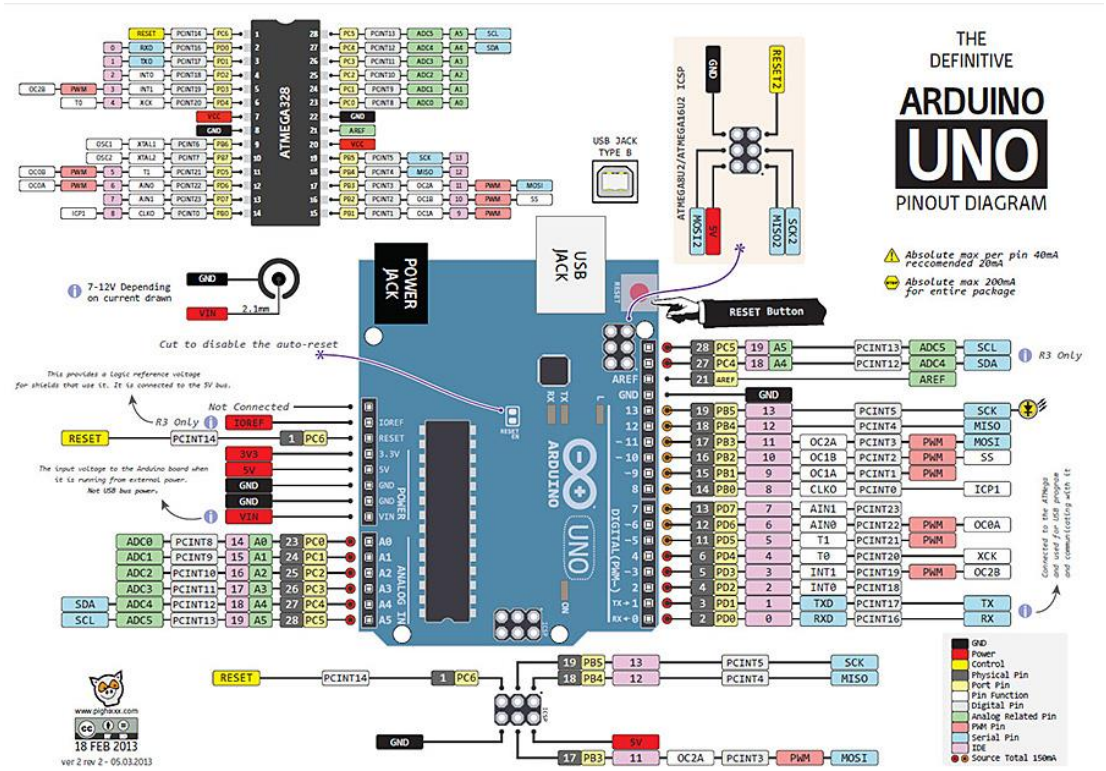
ARDUINO

QUE ES ARDUINO

Arduino es una **placa programable con entradas y salidas digitales y analógicas**, cuyo bajo coste la hace ideal para iniciarse en proyectos de automatización o realizar pequeños proyectos domésticos en electrónica y robótica. Esto significa que disponemos de un pequeño "autómata", capaz de recibir información del entorno (sensores) y realizar acciones (actuadores, motores...), según un programa que introducimos con un ordenador, y que puede ejecutar de forma autónoma.

Arduino es flexible, y fiable. Entonces, ¿Por qué hemos dicho en proyectos domésticos? ¿Por qué no en aplicaciones comerciales o industriales? Es algo que en algún momento os puede tentar, de hecho sabemos que gente saca productos comerciales que están basados en Arduino. Sin embargo, si un día os toca controlar (por ejemplo) la climatización de un polideportivo de 4000m2, donde solo la refrigeradora cuesta 250.000€, os parecerá una temeridad instalar un controlador de 15 euros. Preferiréis instalar un autómata general que cueste 500 a 1000€, que os ofrezca un certificado y una garantía para vuestra tranquilidad, y proteger vuestra firma (aunque allá cada uno con su criterio profesional).

Sin embargo todo lo que hayáis aprendido sobre electrónica, automatización, programación y telecomunicación, lo podréis aplicar directamente y con facilidad en caso de tener que emplear un autómata más caro, por lo que **Arduino es una plataforma excelente para práctica y aprendizaje**. Además os traerá **horas y horas de diversión y entretenimiento** haciendo vuestros proyectos de robótica y electrónica con un bajo coste.



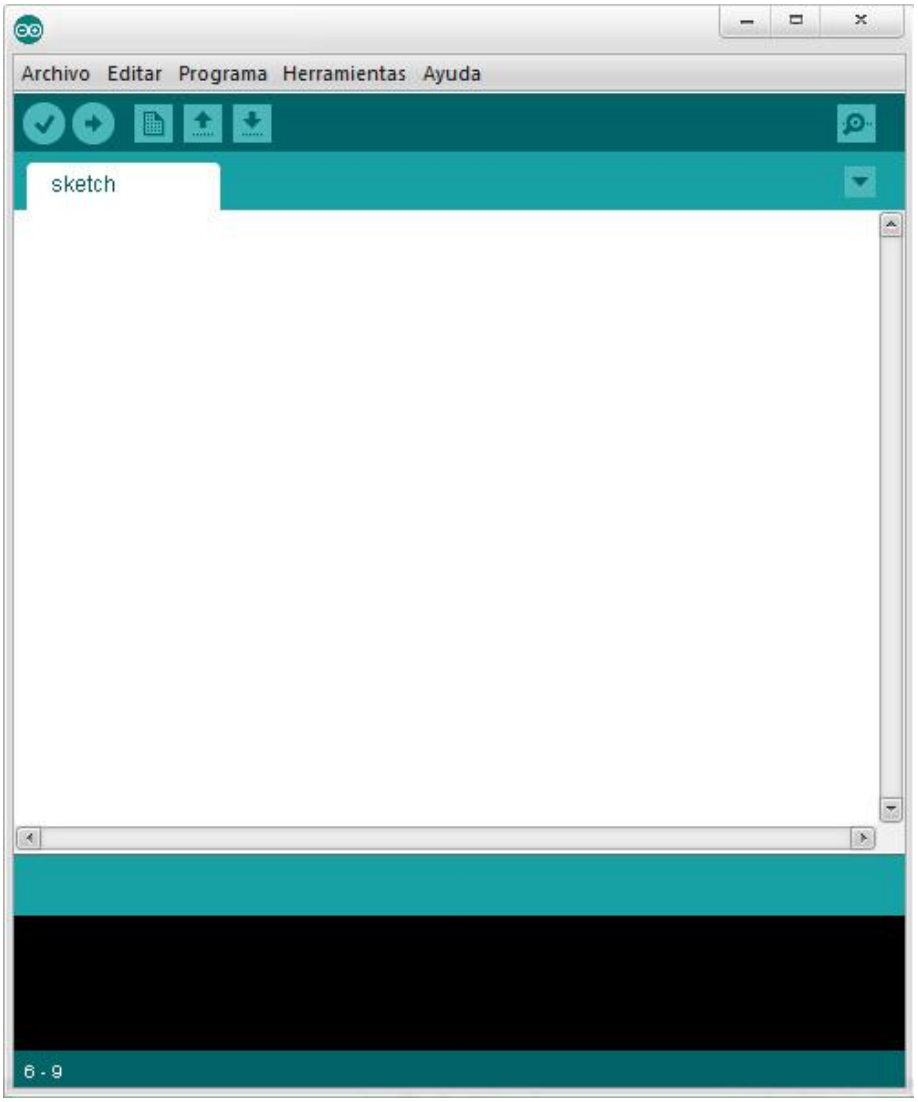
Los modelos y versiones de arduino son muchos y muy variados. Veamos una tabla comparativa de los modelos más famosos:

Fabricante	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino
Modelo	Pro Mini	Nano	Uno	Mega / Mega 2560	Leonardo	Micro	Due
Microcontrolador	AVR Atmega 168 ó 328 8bits	AVR ATmega 168 ó 328 8bits	AVR ATmega 328 8bits	AVR ATmega2560 8bits	AVR ATmega 32u4 8bits	AVR ATmega 32u4 8bits	ARM SAM3X8E Cortex-M3 32bits
Frecuencia	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	84Mhz
Memoria RAM	2KiB	2KiB	2KiB	8KiB	2.5KiB	2.5KiB	96KiB (64+32KiB)
Memoria EEPROM	1KiB	1KiB	1KiB	4KiB	1KiB	1KiB	0
Memoria FLASH	16 ó 32KiB	16 ó 32KiB	32KiB	128 ó 256KiB	32KiB	32KiB	512KiB
Pines digitales entradas/salidas	14/14	14/14	14/14	54/54	20/20	20/20	54/54
Tensión/corriente pines digitales	3.3v ó 5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	3.3v 3~15mA (130mA entre todos)
Pines analógicos entradas/salidas	6/0	8/0	6/0	16/0	12/0	12/0	12/2
Tensión/resolución pines analógicos	3.3v ó 5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	3.3v 12bits (4096 valores)
Pines con interrupción externa	2	2	2	6	2	2	-
Pines PWM	6	6	6	15	7	7	12
Conexiones Serial / UART	1	1	1	4	1	1	4
Conexiones I2C / TWI	1	1	1	1	1	1	2
Conexiones ISP / ICSP	1	1	1	1	1	1	1
Conexión USB	No (necesita adaptador externo)	Si	Si, USB-B	Si, USB-B	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB
Conexión USB de depuración	No	No	No	No	No	No	Si, MicroUSB
Conexión Bluetooth	No	No	No	No	No	No	No
Conexión WiFi	No	No	No	No	No	No	No
Conexión Ethernet	No	No	No	No	No	No	No
Conexión USB Host	No	No	No	No	No	No	Si
Almacenamiento por SD	No	No	No	No	No	No	No
Corriente en el pin de 5v	-	500mA	500~800mA	500~800mA	500~800 mA	500mA	800mA
Corriente en el pin de 3.3v	-	50mA	50mA	50mA	50mA	50mA	800mA
Voltaje de alimentación por el USB	3.3v ó 5v (sin usb)	5v	5v	5v	5v	5v	5v
Voltaje de alimentación recomendado por el Jack	3.35 -12 V (modelo 3.3V) ó 5 - 12 V (modelo 5V)	7~12v	7~12v	7~12v	7~12v	7~12v	7~12v
Voltaje de alimentación limite por el Jack	-	6~20v	6~20v	6~20v	6~20v	6~20v	6~20v
Precio oficial	15+gi	-	20€+gi	40€+gi	18€+gi	18€+gi	39€+gi
Precio BBB	~4€	~9€	~10€	~12€	11€~	~16€	~38€

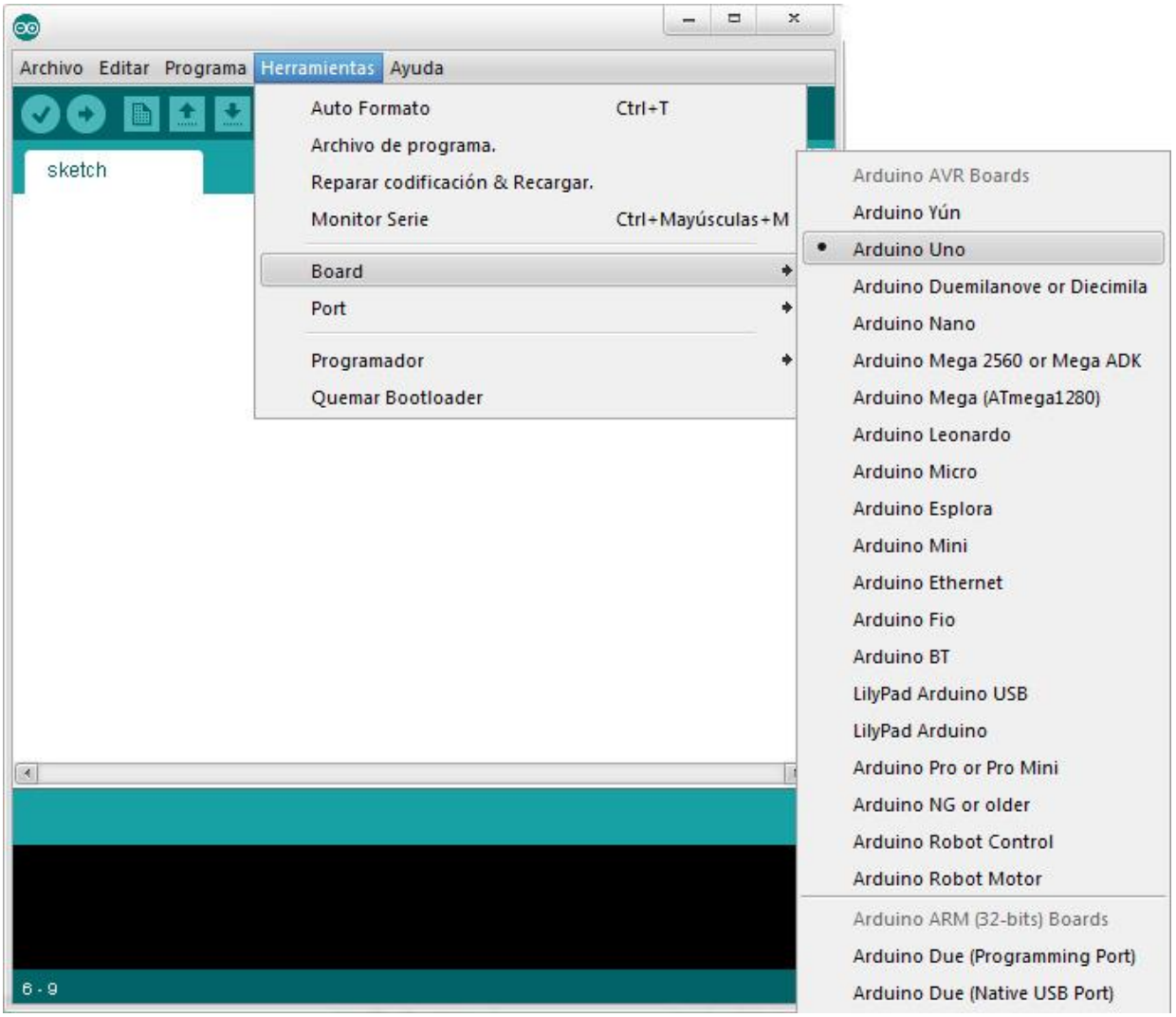
PROGRAMAR CON ARDUINO

En primer lugar, **conectamos nuestra placa Arduino mediante un cable USB A-B**, del tipo que se emplean habitualmente para conectar impresoras. De momento no hace falta la conexión de alimentación o cable adicional, para realizar la programación es suficiente únicamente con el USB.

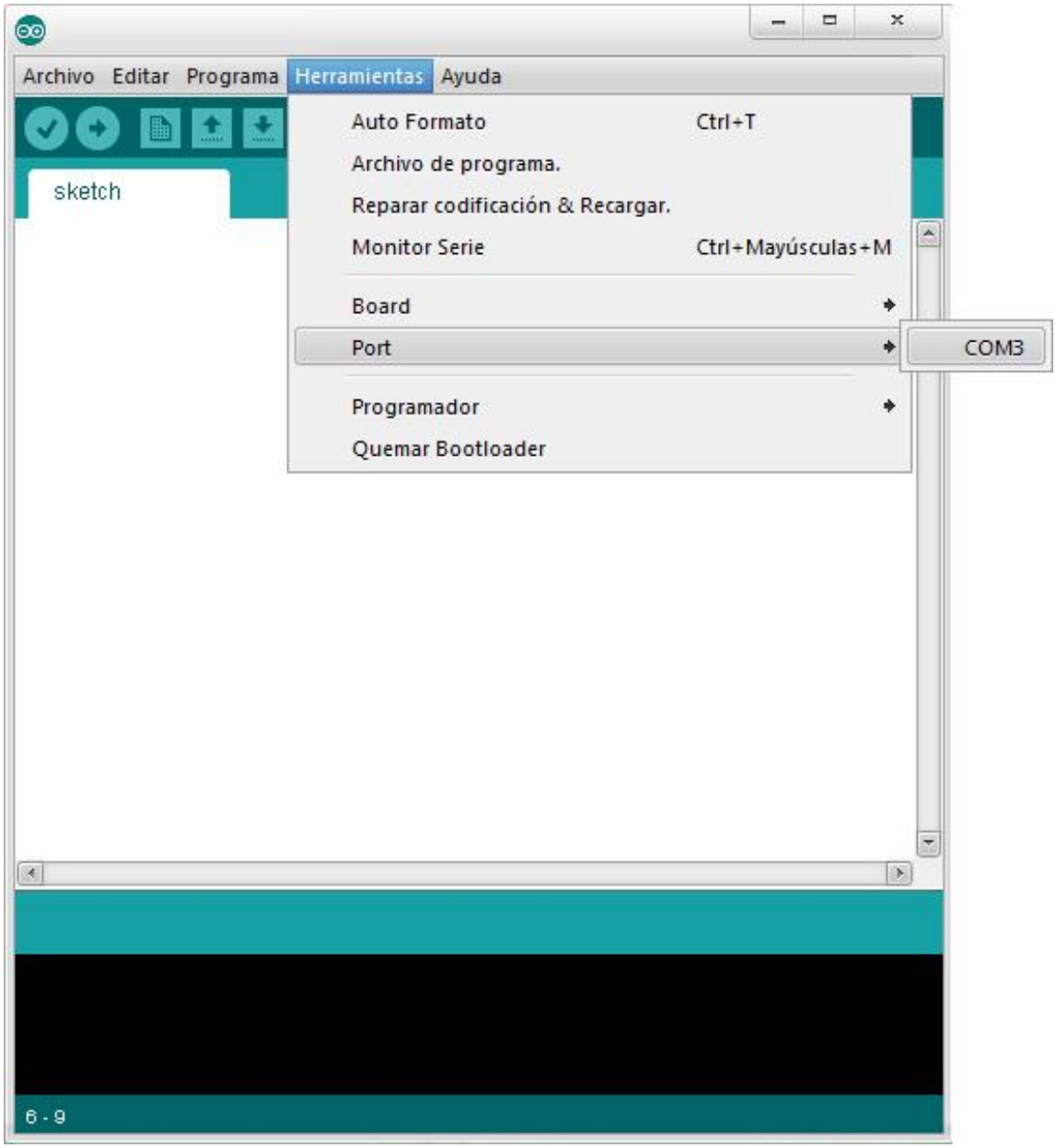
A continuación abrimos el entorno IDE de Arduino.



Seleccionamos el modelo de placa que estemos empleando.

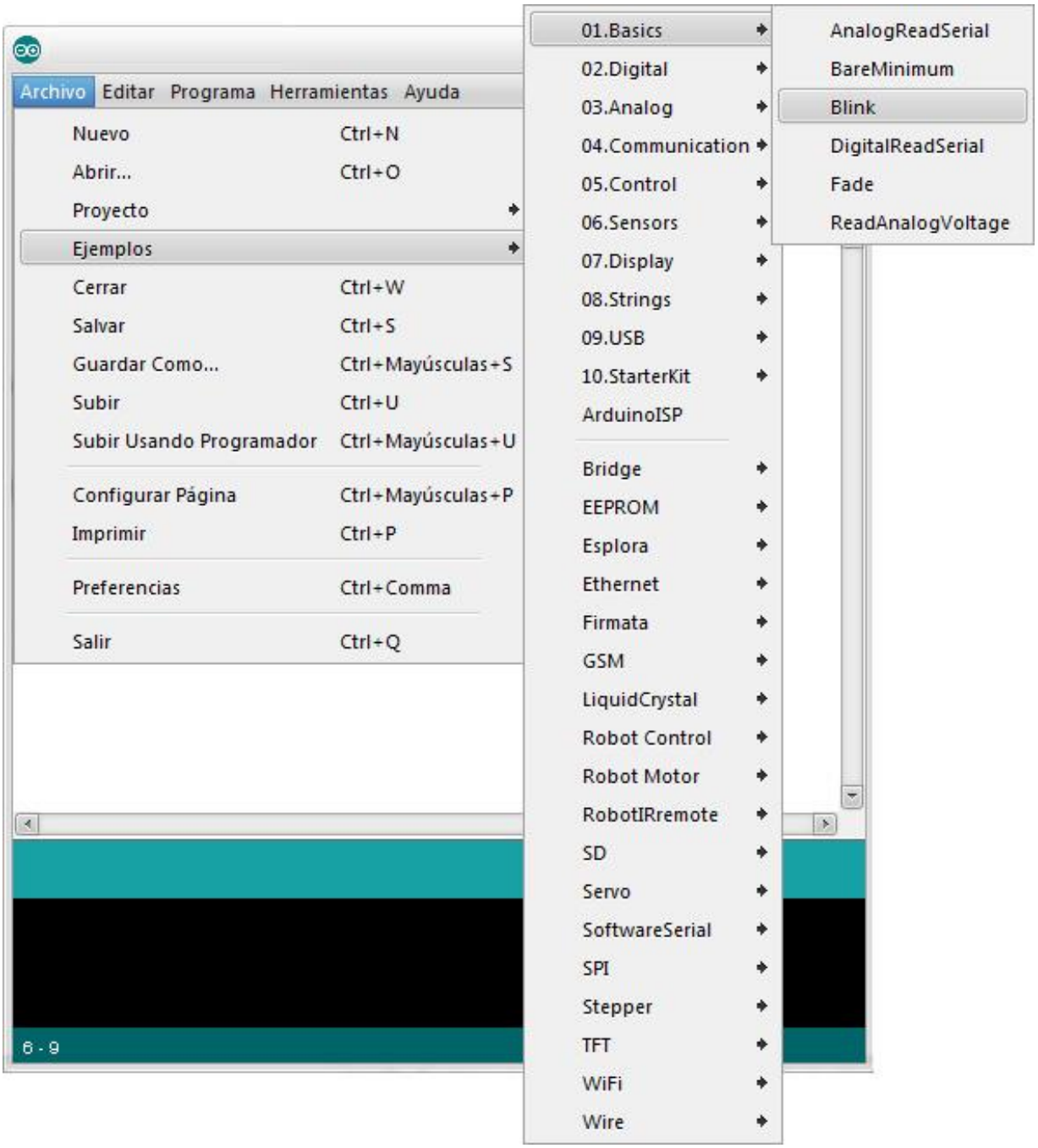


Seleccionamos el puerto de comunicación al que este conectado.



Para probar el funcionamiento de nuestro montaje vamos a emplear uno de los ejemplos incluidos en el IDE de Arduino.

Cargamos el ejemplo "blink" que incluye el IDE de arduino. Este ejemplo representa el "Hola mundo" de la electrónica. Hacer parpadear un led.



Seleccionamos el ejemplo Basics/Blink, y aparecerá un código similar al siguiente.

```
// DECLARACIÓN DE VARIABLES
const int pinLED= 13;            //asignar variable led como 13

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  pinMode(pinLED, OUTPUT);       //definir pin 13 como salida
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
  digitalWrite(pinLED, HIGH);   // encender LED
  delay(1000);                  // esperar un segundo
  digitalWrite(pinLED, LOW);    // apagar LED
  delay(1000);                  // esperar un segundo
}
```

Este ejemplo enciende y apaga un LED cada segundo. La función de cada linea está comentada a la derecha, pero por ahora no nos vamos a preocupar ahora del significado, ya veremos estos aspectos más adelante. El LED empleado está integrado en las placas de Arduino (UNO, MEGA, etc) **conectado físicamente al PIN 13**.

Finalmente, pulsamos en el botón remarcado para compilar y enviar la programación a nuestra placa Arduino. Tras unos segundos, el IDE compilará el programa y la pantalla debería ser similar a la siguiente.

Tras parpadear los leds de TX y RX integrados en la placa, el arduino empezará a ejecutar el programa, encendiendo y apagando el LED. Estamos de acuerdo en que no es muy espectacular que digamos, pero lo importante es que si habéis llegado hasta aquí **significa que todo funciona y está bien configurado**.

COMUNICACIÓN CON PUERTO SERIE

Los puertos serie son la forma principal de comunicar una placa Arduino con un ordenador. Gracias al puerto de serie podemos, por ejemplo, mover el ratón o simular la escritura de un usuario en el teclado, enviar correos con alertas, controlar un robot realizando los cálculos en el ordenador, encender o apagar un dispositivo desde una página Web a través de Internet, o desde una aplicación móvil a través de Bluetooth.

Un puerto serie envía la información mediante una secuencia de bits. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos, RX (recepción) y TX (transmisión). No obstante, pueden existir otros conductores para referencia de tensión, sincronismo de reloj, etc.

Un ordenador convencional dispone de varios puertos de serie. Los más conocidos son el popular USB (universal serial port) y el ya casi olvidado RS-232 (el de los antiguos ratones). Sin embargo, dentro del ámbito de la informática y automatización **existen una gran cantidad adicional de tipos de puertos serie**, como por ejemplo el RS-485, I2C, SPI, Serial Ata, Pcie Express, Ethernet o FireWire, entre otros.

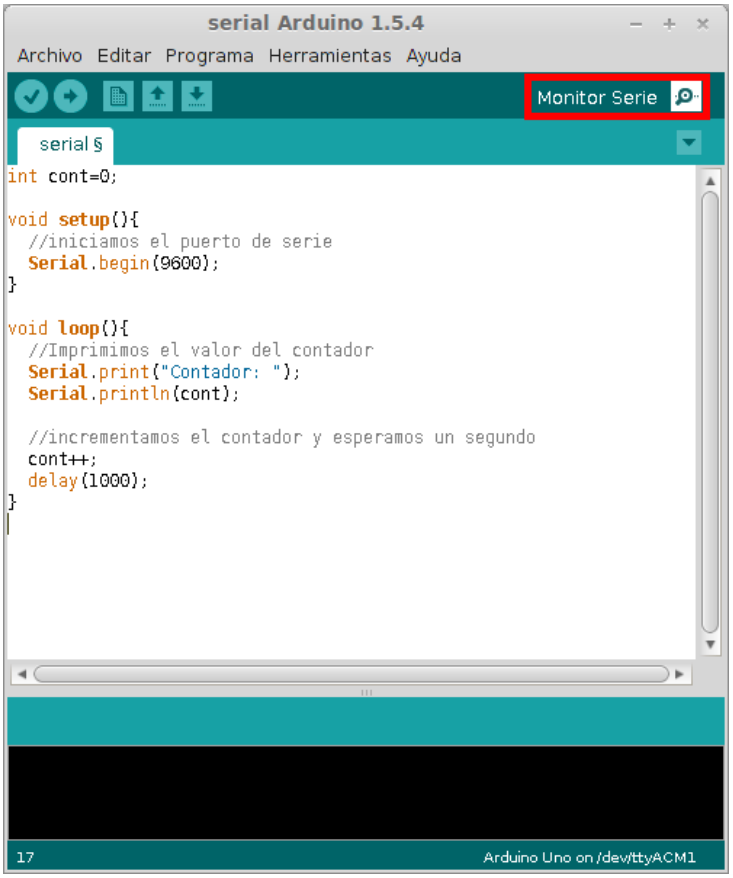
En ocasiones **veréis referirse a los puertos de serie como UART**. La UART (universally asynchronous receiver/transmitter) es una unidad que incorporan ciertos procesadores, encargada de realiza la conversión de los datos a una secuencia de bits y transmitirlos o recibirlos a una velocidad determinada.

Por otro lado, **también podéis oír el término TTL** (transistor-transistor logic). Esto significa que la comunicación se realiza mediante variaciones en la señal entre 0V y Vcc (donde Vcc suele ser 3.3V o 5V). Por el contrario, otros sistemas de transmisión emplean variaciones de voltaje de -Vcc a +Vcc (por ejemplo, los puertos RS-232 típicamente varían entre -13V a 13V).

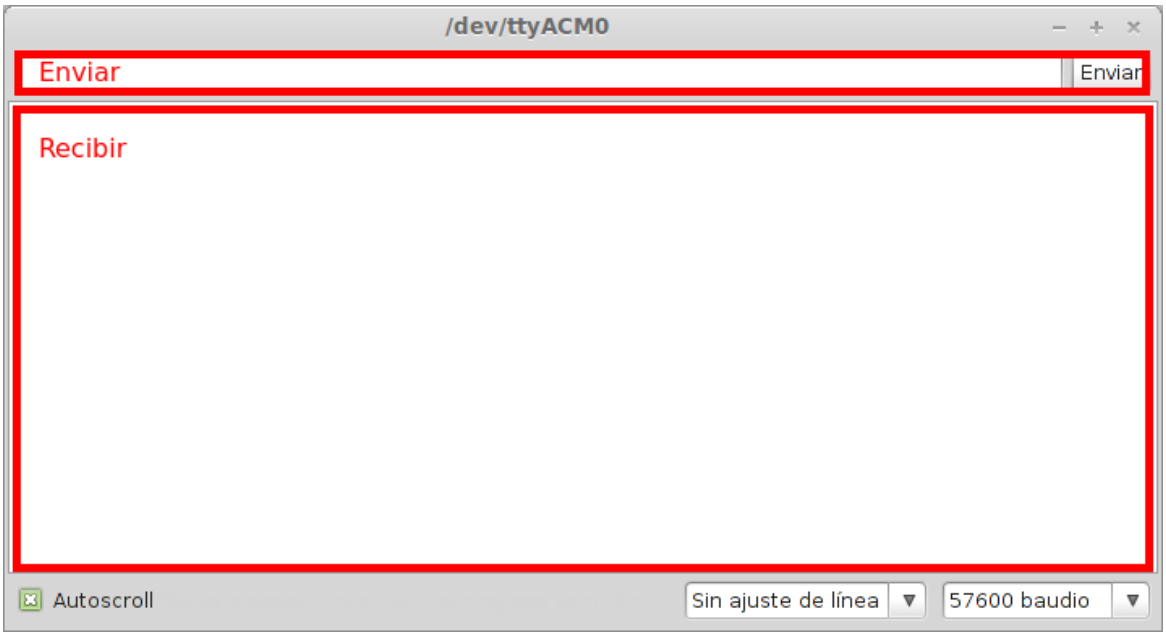
En Arduino UNO y Mini Pro los pines empleados son 0 (RX) y 1 (TX). En el caso de Arduino Mega y Arduino Due el puerto de serie 0 está conectado a los pines 0 (RX) y 1 (TX), el puerto de serie 1 a los pines 19 (RX) y 18 (TX) el puerto de serie 2 a los pines 17 (RX) y 16 (TX), y el puerto serie 3 a los pines 15 (RX) y 14 (TX).

COMO USAR EL PUERTO SERIE

Para realizar la conexión mediante puerto serie únicamente es necesario conectar nuestra placa Arduino empleando el mismo puerto que empleamos para programarlo. A continuación **abrimos el IDE Standard de Arduino y hacemos click en el "Monitor Serial" como se indica en la imagen**.



El monitor de puerto serie es una pequeña utilidad integrada dentro de IDE Standard que **nos permite enviar y recibir fácilmente información a través del puerto serie**. Su uso es muy sencillo, y dispone de dos zonas, una que muestra los datos recibidos, y otra para enviarlos. Estas zonas se muestran en la siguiente imagen.



Pese a su sencillez este monitor de puerto serie es suficiente para los ejemplos de esta entrada, y resulta muy útil para realizar test o experimentos rápidos.

Ejemplo 1: Hola Mundo – Empezando a comunicar por el puerto serie

La configuración del puerto serie se debe realizar en la función setup. Para ello utilizamos el objeto Serial incluido en las librerías Arduino.

Basta que le digas a qué velocidad vas a querer comunicar con el ordenador. Esta velocidad la eliges tú, típicamente toma valores los valores 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, o 115200. La que se suele usar más habitualmente es 9600. Este número indica los bits por segundo a los que se transfiere la información a través del puerto serie. Obviamente, cuanto mayor es este número, comunicaremos a mayor velocidad. Bien, configuremos pues:

Vamos a enviar la cadena de caracteres "**Hola Mundo**" por el puerto serie. Como solo queremos que se envíe una vez lo hacemos en la misma función **setup**:

```
// DECLARACIÓN DE VARIABLES

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
```

```
      Serial.begin(9600); // Iniciamos la comunicación serie a 9600 bits/sec
      Serial.println("Hola Mundo!"); // enviamos "Hola mundo" al puerto serie
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {

}
```

Ejemplo 2: Tic Toc

Ahora vamos a realizar un reloj. Cada segundo escribirá por consola **tic, toc**, para ello programamos la función **loop**, que se ejecutará continuamente

```
// DECLARACIÓN DE VARIABLES

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {

      Serial.begin(9600); //Iniciamos la comunicación serie a 9600 bits/sec
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
      Serial.println("tic"); //escribimos  "tic" en el puerto del monitor serie
      delay(1000); //esperamos 1000 milisegundos
      Serial.println("toc"); //escribimos  "tic" en el puerto del monitor serie
      delay(1000); // esperamos 1000 milisegundos

}
```

Ejemplo 3: Ecoooooooooo

Ahora vamos a recibir datos, y a escribir lo mismo que hemos recibido:

La función **Serial.available()** del objeto Serial devuelve el número de bytes disponibles en el puerto serie (para ser leídos). Por lo tanto, cuando hay datos disponibles (**Serial.available()>0**) los leemos (**Serial.read()**) y lo mismo que estamos leyendo lo escribimos por el puerto serie (**Serial.write()**). Dado a que no estamos incluyendo el cambio de línea automática el echo es seguido.

```
// DECLARACIÓN DE VARIABLES

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
      Serial.begin(9600); // Iniciamos la comunicación serie a 9600 bits/sec
}
```

```
// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
      if (Serial.available()>0){ //Si hay datos en el buffer del puerto serie
      Serial.write(Serial.read()); //Escribimos en el puerto serie lo enviado
}
```

Posibles prácticas recomendadas:

1. Enviar información de una variable contador desde arduino cada segundo
2. Recibir información para encender o apagar el led 13
3. Cambiar la velocidad de parpadeo de un LED desde el monitor serie

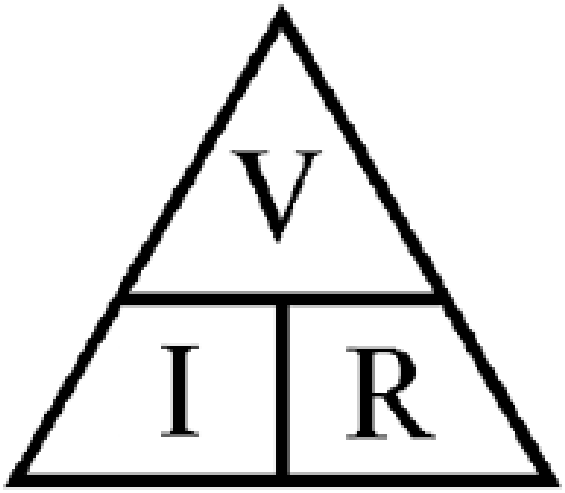
VOLTAJE, CORRIENTE Y LA LEY DE OHM

El **Voltaje** es una mágnitud física que impulsa a los electrones a lo largo de un conductor (por ejemplo un cable) en un circuito eléctrico cerrado, provocando el flujo de una corriente eléctrica. Su unidad es el Voltio(V). El instrumento usado para medir el voltaje se conoce como voltímetro.

La **corriente (o intensidad)** es el flujo de electrones a través de un conductor o semiconductor en un sentido. La unidad de medida de ésta es el amperio (A). Una corriente eléctrica, puesto que se trata de un movimiento de cargas, produce un campo magnético, un fenómeno que puede aprovecharse en el electroimán, este es el principio de funcionamiento de un motor. El instrumento usado para medir la intensidad de la corriente eléctrica es el galvanómetro que, calibrado en amperios, se llama amperímetro, colocado en serie con el conductor cuya intensidad se desea medir.

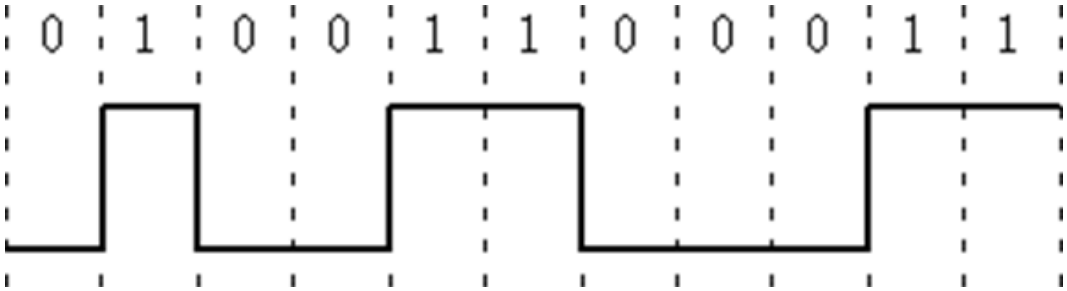
La **LEY DE OHM** dice que la corriente (I) que circula por un conductor eléctrico es directamente proporcional al voltaje (V) e inversamente proporcional a la resistencia (R).

La pirámide de la imagen siguiente es muy útil para conocer la fórmula a la que es igual la variable que tapes con el dedo, por ejemplo: Tapa con tu dedo la V (voltaje), entonces voltaje va a ser igual a I (corriente) por R (resistencia), una más, tapa I (Corriente), I va ser igual a V divido R.

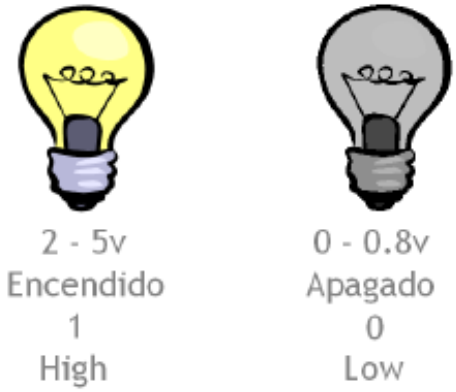


ENTRADAS Y SALIDAS DIGITALES

Una señal digital es una **variación de voltaje entre -Vcc a +Vcc sin pasar por los valores intermedios**. Por lo tanto, una señal digital dispone solo de dos estados. Al valor inferior de tensión –Vcc (o GND) le asociamos un valor lógico LOW o '0', mientras que al valor superior +Vcc le asociamos HIGH o '1' lógico.



En Arduino las entradas y salidas digitales comparten pin y corresponden a los pines que van del **0 al 13 en Arduino UNO y leonardo** y del **0 al 53 en Arduino Mega y DUE**, motivo por el que se denominan I/O digitales. Esto significa que el mismo pin puede ejecutar funciones tanto de entrada como de salida aunque, lógicamente, no de forma simultánea. Es necesario configurar un pin I/O como entrada o salida en el código.



CÓDIGO PARA ENTRADA DIGITAL

El código para realizar la lectura es realmente sencillo. Simplemente tenemos que configurar un I/O digital como entrada con **pinMode(pin, INPUT)** y realizar la lectura con **digitalRead(pin)**.

```
// DECLARACIÓN DE VARIABLES
int pin = 2;
int value = 0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  Serial.begin(9600);   //iniciar puerto serie
  pinMode(pin, INPUT);  //definir pin como entrada
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
  value = digitalRead(pin);  //lectura digital de pin

  //mandar mensaje a puerto serie en función del valor leido
  if (value == HIGH) {
      Serial.println("Encendido");
  }
  else {
      Serial.println("Apagado");
  }
  delay(1000);
}
```

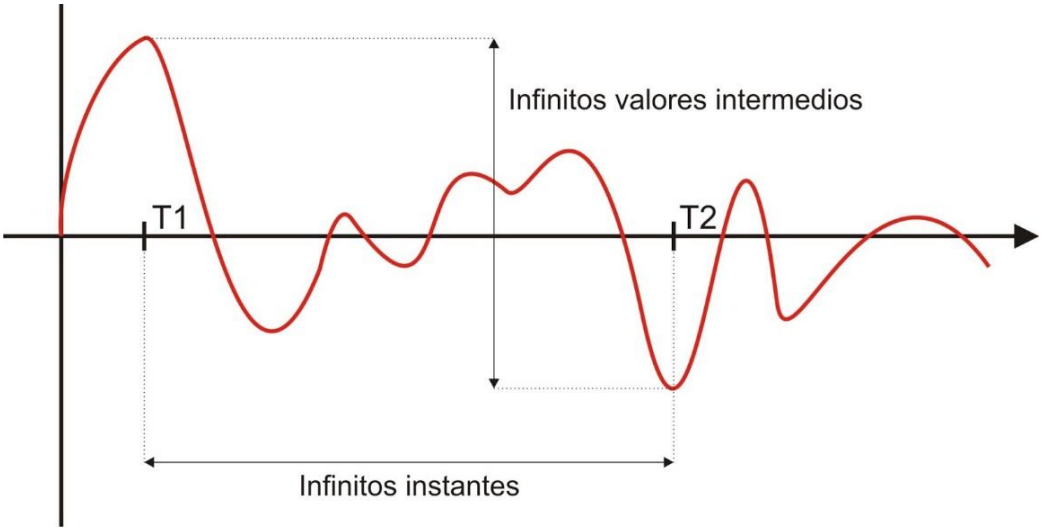
CÓDIGO PARA SALIDA DIGITAL

El código para escribir en un pin es realmente sencillo. Simplemente tenemos que configurar un I/O digital como salida con **pinMode(pin,OUTPUT)** y realizar la escritura con **digitalWrite(pin, HIGH o LOW)**.

```
// DECLARACIÓN DE VARIABLES
int pin = 13;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  pinMode(pin, OUTPUT);  //definir pin como salida
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
  digitalWrite(pin, HIGH);   // poner el Pin en HIGH
  delay(1000);               // esperar un segundo
  digitalWrite(pin, LOW);    // poner el Pin en LOW
  delay(1000);               // esperar un segundo
}
```

ENTRADAS Y SALIDAS ANALÓGICAS

Una señal analógica es una magnitud que **puede tomar cualquier valor dentro de un intervalo –Vcc y + Vcc**. Por ejemplo, una señal analógica de tensión entre 0V y 5V podría valer 2,72V, o cualquier otro valor con cualquier número de decimales.



Por norma general, en los autómatas **las entradas analógicas son más escasas, más lentas y más caras** que las entradas digitales. En el caso de Arduino uno disponemos de un número variable de entradas analógicas, que en el caso de Arduino Uno y Mini Pro son 6, en el Arduino leonardo pueden ser hasta 10 (alguna digital se puede usar como analógica en esta placa) y en el Arduino Mega y DUE son 16. Este es un número más que respetable de entradas analógicas, que rivaliza o supera a autómatas tradicionales de coste muy superior.

CÓDIGO PARA ENTRADA ANALÓGICA

El código para realizar la lectura es realmente sencillo, y similar al que vimos para las entradas digitales. Simplemente realizamos la lectura mediante **AnalogRead(pin)** y almacenamos el valor devuelto.

El valor devuelto es un número entero entre 0 a 1023. En caso de querer convertir este valor en un valor de tensión podemos hacerlo de la siguiente manera:

```
// DECLARACIÓN DE VARIABLES
const int sensorPin = A0;        // seleccionar la entrada para el sensor
int sensorValue;                 // variable que almacena el valor raw (0 a 1023)
float value;                     // variable que almacena el voltaje (0.0 a 5.0)
```

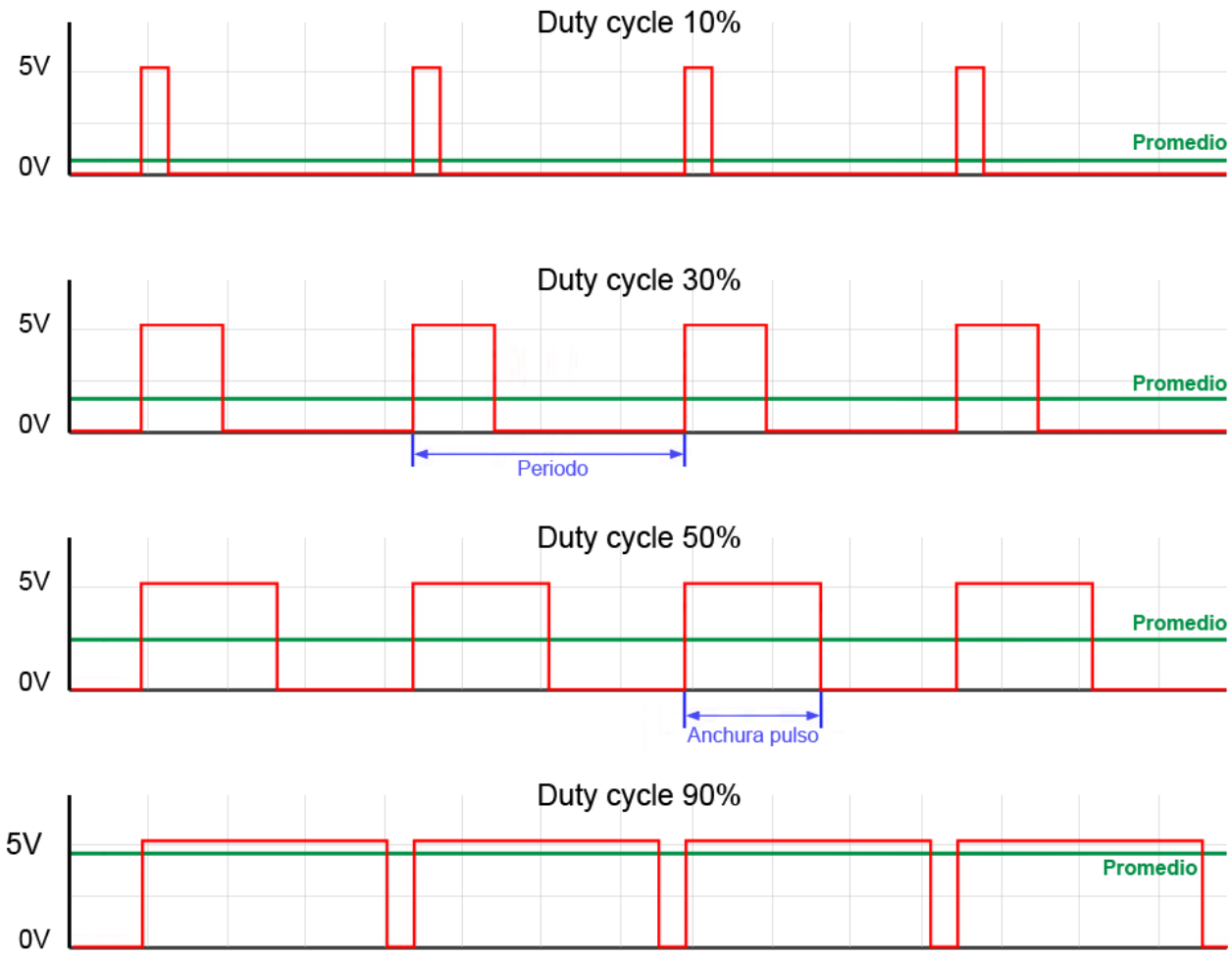
```
// FUNCIONES
// función de cambio de escala entre floats
float fmap(float x, float in_min, float in_max, float out_min, float out_max)
{
      return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}


// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
             sensorValue = analogRead(sensorPin);              // realizar la lectura
      value = fmap(sensorValue, 0, 1023, 0.0, 5.0);   // cambiar escala a 0.0 - 5.0

      Serial.println(value);                        // mostrar el valor por serial
      delay(1000);
}
```

SALIDAS ANALÓGICAS

Las salidas analógicas, por el contrario, son algo más complicadas que las digitales.

Lo primero que tenemos que entender es que la mayoría de automatismos (y Arduino no es una excepción) **no son capaces de proporcionar una auténtica salida analógica**.

Para salvar esta limitación y simular una salida analógica la mayoría de los automatismos emplean un "truco", que consiste en **activar una salida digital durante un tiempo y mantenerla apagada durante el resto**. El promedio de la tensión de salida, a lo largo del tiempo, será igual al valor analógico deseado.



La proporción de tiempo que está encendida la señal, respecto al total del ciclo, se denomina "Duty cycle", y generalmente se expresa en tanto por ciento.

Es inmediato deducir que la señal promedio es el producto de la tensión máxima y el DutyCycle, según la siguiente expresión.

$$V_{medio} = (V_{cc+} - V_{cc-}) \cdot \frac{DutyCycle}{100}$$

Es importante recordar en todo momento que **en una salida PWM el valor de tensión realmente es Vcc**. Por ejemplo, si estamos alimentando un dispositivo que necesita 3V, y usamos una pulsada, en realidad estaremos suministrando 5V durante un 60% del tiempo y 0V durante el 40%. Pero si el dispositivo, por ejemplo, soporta como máximo 3V, **podemos dañarlo si lo alimentamos mediante un PWM**.

Arduino implementa por hardware salidas PWM en varios de sus pines, que aparecen identificados en la placa con el símbolo "~" junto al número del pin. También podemos emular por software señales PWM, pero con la carga de trabajo adicional que ello supone para el procesador.

- En **Arduino Uno, Mini y Nano,** disponemos de 6 salidas PWM de 8bits en los pines 3, 5, 6, 9, 10 y 11.
- En **Arduino Mega** disponemos de 15 salidas PWM de 8bis en los pines 2 a 13 y 44 a 46
- **Arduino Due** cuenta con 13 salidas PWM de 8bits en los pins 2 a 13. Además, esta placa incorpora dos salidas analogicas discretizadas (DAC) con resolución de 12bits (4096 niveles)

Una resolución de 8bits en una salida PWM significa que tenemos 256 niveles. Es decir, indicamos el Duty cycle mediante un número de 0 a 255.

CÓDIGO PARA SALIDAS ANALÓGICAS

El código necesario para encender una salida PWM es muy sencillo gracias a las bibliotecas de Arduino, que configuran por defecto las salidas de PWM en la función Setup, ocultando la dificultad de manipulación de los Timer.

Así, en en el ejemplo mas básico, simplemente definimos el pin PWM que queremos emplear, y usamos la función **analogWrite(pin, valor)** para escribir el valor del Duty Cycle, medido de 0 a 255.

```
// DECLARACIÓN DE VARIABLES
const int analogOutPin = 11; // pin de salida analógico
byte outputValue = 0;        // valor del PWM

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {

}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
      analogWrite(analogOutPin, outputValue);
      delay(10);
      outputValue++;
}
```

PRÁCTICAS BÁSICAS

LEER UN PULSADOR CON ARDUINO

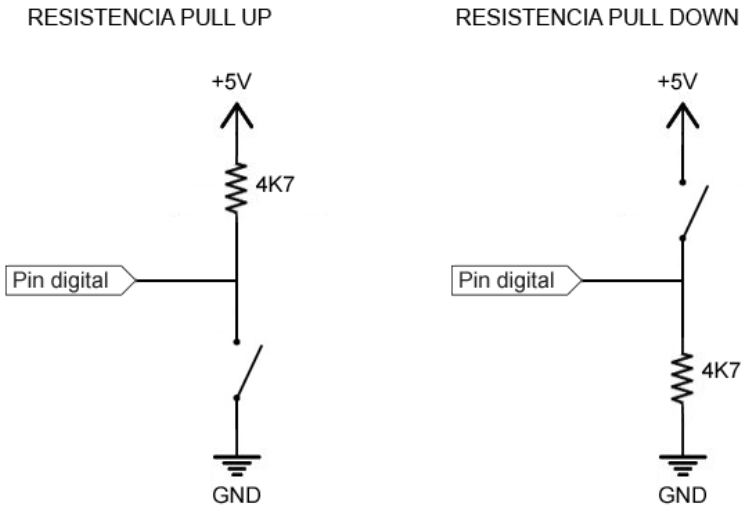
Para la realización de esta práctica necesitamos:

1- Placa controladora Arduino
2- Cables dupont
3- Resistencia 4K7Ohm
4- Pulsador

Para el montaje correcto de cualquier tipo de interruptor en arduino debemos usar una resistencia de alguna de estas dos configuraciones:

1- Resistencia en Pull Down
2- Resistencia en Pull Up

Estas dos resistencias son un mecanismo básico, muy habitual dentro del mundo de la electrónica y automatización.

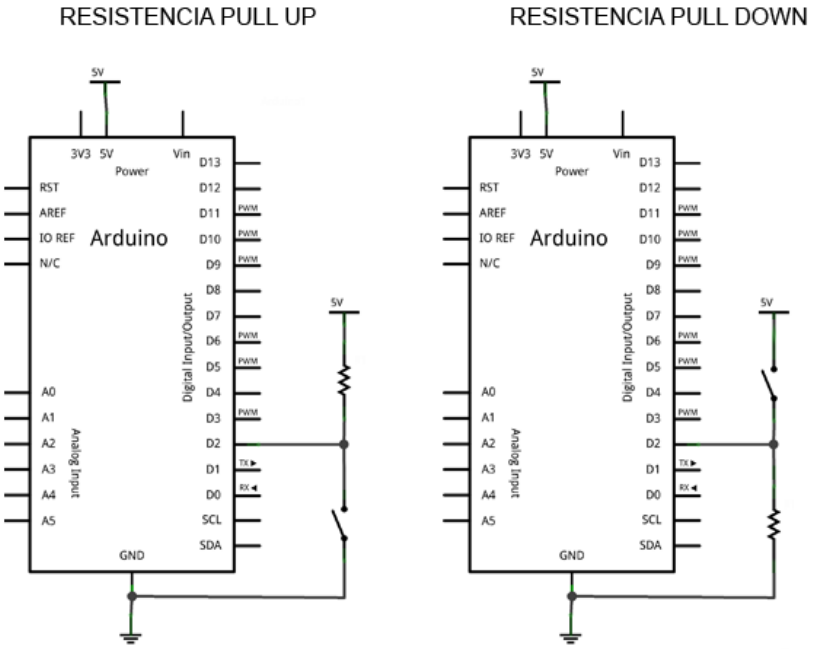


Las resistencias de Pull-Down y Pull-Up **se conectan entre el PIN digital y una de las tensiones de referencia** (0V o 5V) y "fuerzan" (de ahí su nombre) el valor de la tensión a LOW o HIGH, respectivamente.

- **La resistencia de Pull-Up fuerza HIGH cuando el pulsador está abierto**. Cuando está cerrado el PIN se pone a LOW, la intensidad que circula se ve limitada por esta resistencia

- **La resistencia de Pull-Down fuerza LOW cuando el pulsador está abierto.** Cuando está cerrado el PIN se pone a HIGH, y la intensidad que circula se ve limitada por esta resistencia

Así es como quedaría el montaje final en vista esquemática (la conexión de puede realizar empleando cualquiera de los PIN digitales.



Por último, el código de lectura del estado del PIN es igual que vimos anteriormente en las entradas digitales.

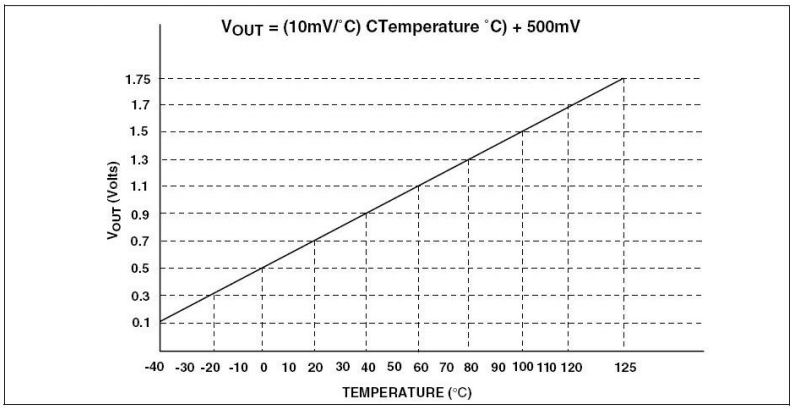
```
// DECLARACIÓN DE VARIABLES
const int inputPin = 2;
int value = 0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  Serial.begin(9600);
  pinMode(inputPin, INPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
  value = digitalRead(inputPin);  //lectura digital de pin
  //mandar mensaje a puerto serie en función del valor leido
  if (value == HIGH) {
      Serial.println("Encendido");
  }
  else {
      Serial.println("Apagado");
  }
  delay(1000);
}
```

Arduino dispone de resistencias de Pull-Up internas de 30k pero no suelen ser usadas por dos motivos.

- Tienen poca autoridad (valor de resistencia demasiado alto)
- Si la configuramos incorrectamente desde el programa, podemos generar un cortocircuto, por lo que es preferible conectarla físicamente para verificar que no nos la hemos dejado.

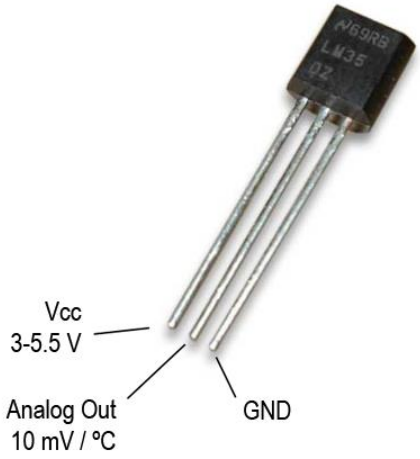
LEER EL VALOR DE TEMPERATURA DEL LM35DZ

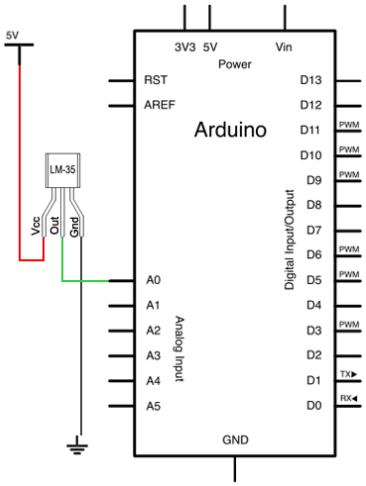
La salida del LM35 es lineal con la temperatura, incrementando el valor a razón de 10mV por cada grado centígrado. El rango de medición es de -55ºC (-550mV) a 150ºC (1500 mV). Su precisión a temperatura ambiente es de 0,5ºC.



El patillaje del LM35 se muestra en la siguiente imagen. Los pines extremos son para alimentación, mientras que el pin central proporciona la medición en una referencia de tensión, a razón de 10mV/ºC.

El esquema eléctrico de montaje para este componente es muy sencillo:





El código necesario para realizar la lectura es también muy simple. Simplemente leemos el valor del tensión mediante una de las entradas analógica, y traducimos el valor a grados celsius usando la relación 10 mV/C.

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
```

```
const int sensorPin= A0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
Serial.begin(9600);                // inicializamos la comunicación Serie
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  int value = analogRead(sensorPin);
  float volts = (value / 1024.0) * 5;
  float celsius = (volts - 0.5 ) * 100;
  Serial.print("La temperatura es:");
  Serial.println(celsius);
  delay(1000);
}
```

LEER EL VALOR DEL SENSOR DE AGUA

Es funcionamiento de estos sensores son muy sencillos. Una pcb con pistas al aire que se cortocircuitas al entrar en contacto con el agua variando de ese modo la resistividad del componente.

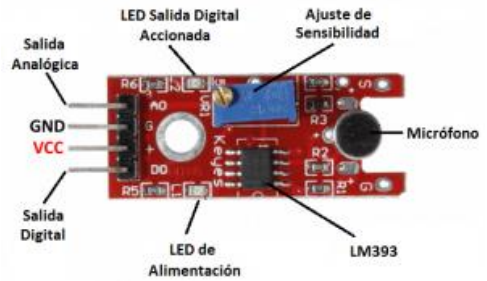
Debemos tener en cuenta que estos componentes no ofrecen una gran precisión en su medida. Es preferible utilizarles como interruptores de agua/no-agua con un umbral de detección.

Su montaje es equivalente al sensor LM35. Dispone de 3 pines: Vcc, GND y Señal.

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int sensorPin= A0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
Serial.begin(9600);                // inicializamos la comunicación Serie
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  int value = analogRead(sensorPin);
  Serial.print("SensorValor es:");
  Serial.println(value);
  delay(20);  // siempre es recomendable dejar un pequeño tiempo entre lecturas del
conversor analógico-digital
}
```

LEER EL VALOR DEL SENSOR DE SONIDO

Este tipo de sensores consta de un micrófono soldado en una pequeña placa PCB que nos facilita el montaje.



En la parte de la izquierda vemos lo pines de conexión:

- En el centro tenemos la conexión a **5V** y a **GND** *(+ y G).*
- **D0** es una salida digital que actúa a modo de **comparador**. Si el sonido captado por el micrófono supera un determinado nivel se pone a HIGH.
- **A0** es una salida analógica que nos da un valor entre 0 y 1023 en función del volumen del sonido.

Además tenemos dos LEDs, uno que nos indica si el sensor está alimentado y otro que se ilumina si D0 está a HIGH.

El ajuste de sensibilidad del micrófono lo hacemos mediante un potenciómetro que tendremos que girar con un destornillador plano.

En esta práctica, vamos a aprender a calibrar el sensor. Conectaremos el display LCD con I2C para mostrar los valores detectados y los pines analógicos y digitales del sensor a los pines que queramos del arduino.

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int sensorPin= A0;
const int sensorPin_dig=7;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
Serial.begin(9600);                // inicializamos la comunicación Serie
pinMode(sensorPin_dig, INPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  int value = analogRead(sensorPin);
  int value_dig = digitalRead(sensorPin_dig);
  Serial.print("SensorValor es:");
  Serial.print(value);
  Serial.print(value_dig);
  if (value_dig == 1){
    delay(1000);
```

```
  }
  delay(1);  // siempre es recomendable dejar un pequeño tiempo entre lecturas del
conversor analógico-digital
}
```

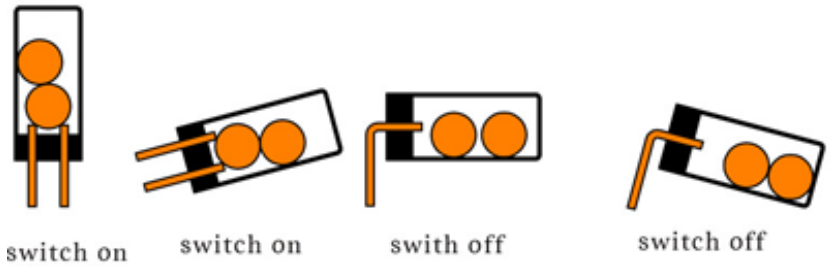
Posibles prácticas recomendadas:

1. Hacer encender un led con dos palmadas
2. Hacer un sonómetro con caracteres en el display en lugar de números

LEER EL VALOR DEL SENSOR DE INCLINACIÓN TILT

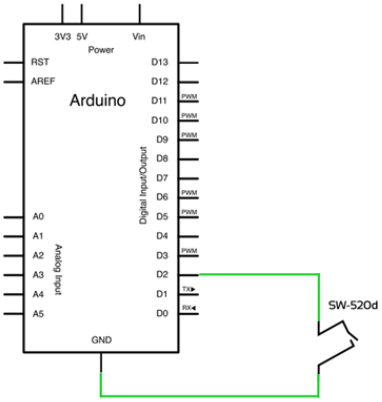
Los sensores tilt son sensores que funcionan como un interruptor cerrando o abriendo el circuito cuando la inclinación supere un cierto umbral.

Su funcionamiento es muy sencillo. Se dispone de un cilindro cuya pared constituye un contacto eléctrico, mientras que el otro contacto está localizado en el centro de la base. Al inclinar lo suficiente el dispositivo ambas esferas constituyen un puente entre ambos contactos, cerrando el circuito.



Debido a su principio de funcionamiento, estos sensores resultan sensibles a movimientos bruscos y vibraciones.

En este caso, y para evitar el uso de resistencias externas utilizaremos la resistencia pull-up internet en el arduino. Para ello, conectaremos el sesor tilt de la siguiente manera:




```
// DECLARACIÓN DE VARIABLES
const int SensorPin = 12;
const int LEDPin = 13;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
    Serial.begin(9600);                // inicializamos la comunicación Serie
    pinMode(SensorPin , INPUT_PULLUP); //activamos la resistencia interna PULL UP
    pinMode(LEDPin, OUTPUT);
}
```

```
// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
    if (digitalRead(SensorPin)) { //encendemos/apagamos el LED cuando movemos el
sensor TILT
        digitalWrite(LEDPin, HIGH);
        Serial.println("movimiento detectado");
    }  else {
        digitalWrite(LEDPin, LOW);
    }
}
```

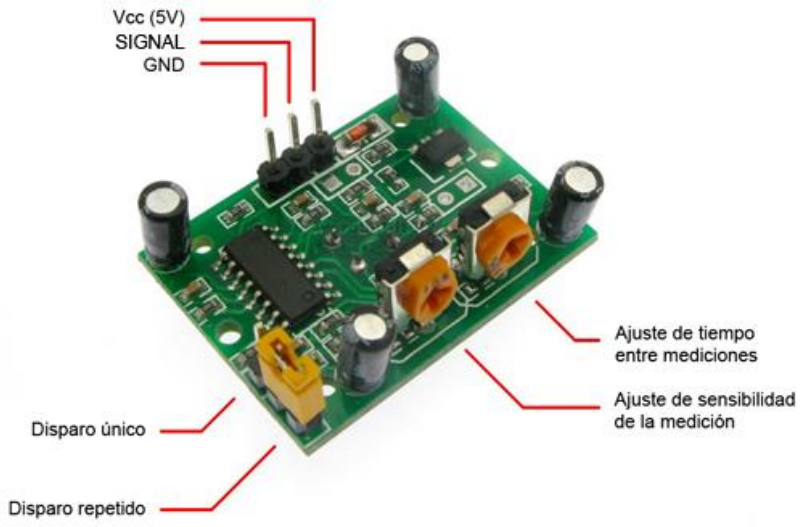
Posibles prácticas recomendadas:

1. Hacer una bola mágica que responda **si** o **no** después de moverla
2. Hacer un reloj de arena con 5 leds que se reinicie al darle la vuelta

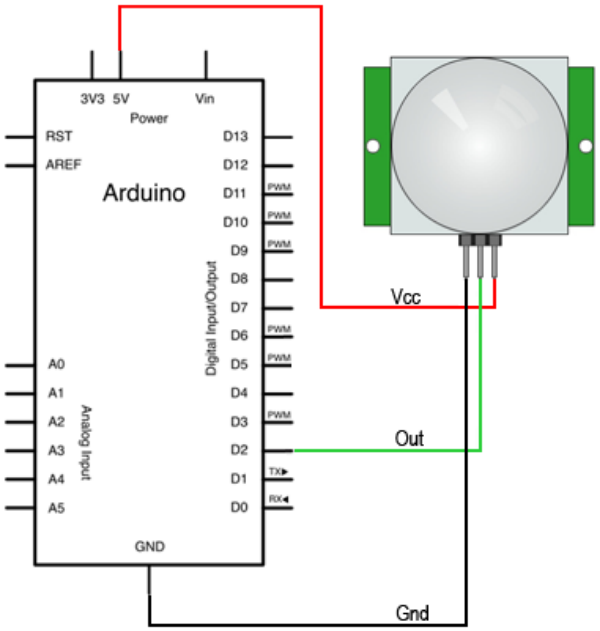
LEER EL VALOR DEL SENSOR DE MOVIMIENTO PIR

Los sensores PIR **se basan en la medición de la radiación infrarroja**. Todos los cuerpos (vivos o no) emiten una cierta cantidad de energía infrarroja, mayor cuanto mayor es su temperatura. Los dispositivos PIR disponen de un sensor piezo eléctrico capaz de captar esta radiación y convertirla en una señal eléctrica.

Este es el esquema de patillaje de un sensor PIR.



El esquema eléctrico que necesitamos es el siguiente.



En esta práctica vamos a mostrar el texto "sensor activado / sensor parado" en el display LCD cuando el sensor detecte algo.

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int LEDPin = 13;        // pin para el LED
const int PIRPin = 2;         // pin de entrada (for PIR sensor)

int pirState = LOW;           // de inicio no hay movimiento
int val = 0;                  // estado del pin

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);                // inicializamos la comunicación Serie
  pinMode(LEDPin, OUTPUT);
  pinMode(PIRPin, INPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  val = digitalRead(PIRPin);
  if (val == HIGH)   //si está activado
  {
    digitalWrite(LEDPin, HIGH);  //LED ON
    if (pirState == LOW)  //si previamente estaba apagado
    {
      Serial.println("Sensor activado");
      pirState = HIGH;
    }
  }
  else   //si esta desactivado
  {
    digitalWrite(LEDPin, LOW); // LED OFF
    if (pirState == HIGH)  //si previamente estaba encendido
    {
      pirState = LOW;
    }
  }
}
```

LEER VALORES DEL JOYSTICK PS3

Estos joystick son similares a las palancas de analógicas que tienen muchos mandos de videoconsolas u ordenadores, como el mando de Playstation 2, o Xbox, que se emplean cuando se necesita un control más suave y preciso del que es posible con los mandos digitales.

Internamente los joystick están formados por un sistema de balancín con dos ejes ortogonales acoplados a dos potenciómetros. Estos potenciómetros realizan la medición de la posición de la palanca en ambos ejes.

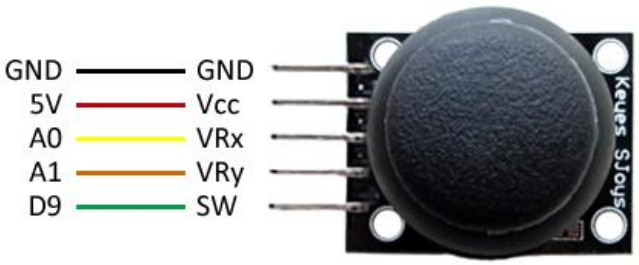
Por otro lado, uno de los ejes está apoyado en un microrruptor, lo que permite detectar la pulsación de la palanca. Este tipo de joystick son dispositivos sencillos y cómodos de usar, y nos permiten añadir un control a nuestros proyectos. Por ejemplo, podemos usarlo para controlar un servo, un brazo robótico, una torreta con un laser, o incluso vehículos y robots completos.

El montaje de este tipo de dispositivos es sencillo. En primer lugar, alimentamos el módulo conectando Vcc y GND, respectivamente, a 5V y GND de Arduino.

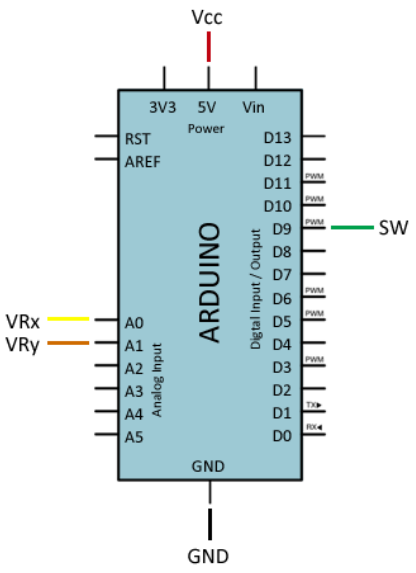
Para la medida analógica en ambos ejes, conectamos las salidas VRx y VRy a dos entradas analógicas de Arduino.

Finalmente, si también queremos la señal digital del pulsador, conectamos la salida SW a una entrada digital de Arduino. Vamos a usas las resistencias de Pull-Up internas de Arduino, por lo que no es necesario resistencias adicionales.

El esquema de montaje del joystick quedaría de la siguiente forma



Mientras que el montaje, visto desde Arduino, sería el siguiente.




```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int pinJoyX = A0;
const int pinJoyY = A1;
const int pinJoyButton = 9;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);                 // inicializamos la comunicación Serie
  pinMode(pinJoyButton, INPUT_PULLUP); // activamos la resistencia pull-up interna
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  int Xvalue = 0;
  int Yvalue = 0;
  bool buttonValue = false;

  //leer valores
  Xvalue = analogRead(pinJoyX);
  delay(100);           //es necesaria una pequeña pausa entre lecturas analógicas
  Yvalue = analogRead(pinJoyY);
  buttonValue = digitalRead(pinJoyButton);

  //mostrar valores por serial
  Serial.print("X:" );
  Serial.print(Xvalue);
  Serial.print("| Y: ");
  Serial.print(Yvalue);
  Serial.print("Pulsador: ");
  Serial.println(buttonValue);
  delay(250);
}
```

ACCIONAR RELE 220V/10A

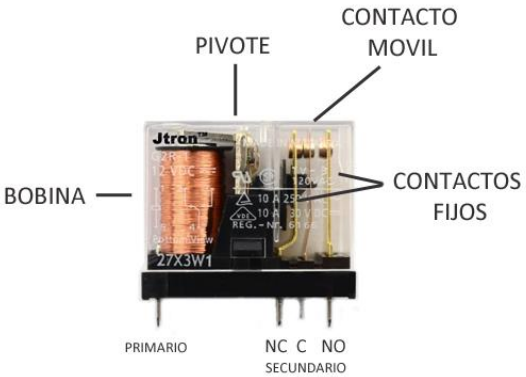
Las salidas por relé son muy frecuentes en el campo de la automatización de procesos, y casi todos los autómatas incluyen salidas por relé para accionar cargas como motores, bombas, climatizadores, iluminación, o cualquier otro tipo de instalación o maquinaria.

Físicamente **un relé se comporta como un interruptor "convencional"** pero que, en lugar de accionarse manualmente, es activado de forma electrónica. Los relés son aptos para accionar cargas tanto de corriente alterna como continua.

Un relé dispone de dos circuitos:

- **El circuito primario** se conecta con la electrónica de baja tensión, en nuestro caso Arduino, y recibe la señal de encendido y apagado.
- **El circuito secundario** es el interruptor encargado de encender o apagar la carga.

Al ser dispositivos electromecánicos que requieren el movimiento de componentes interno para su funcionamiento **el tiempo de conmutación de un relé es elevado**, del orden de 10ms.



Los relés **normalmente disponen de tres contactos en el secundario** C (común), NO (normalmente abierto) y NC (normalmente cerrado). Pero también encontramos modelos que prescinden del terminal NC.

Cuando se activa el relé, **la corriente circula por la bobina del circuito primario generando un campo magnético** que hace pivotar una armadura, que a su vez empuja al contacto móvil, cerrando el circuito con el contacto fijo NO. Mientras, se separa y abre el circuito con el terminal NC.

- **El terminal NO** (normalmente abierto) está aislado de C cuando el relé está apagado y conectado cuando el relé está encendido.
- **El terminal NC** (normalmente cerrado) está conectado con C cuando el relé está apagado, y aislado cuando el relé está encendido.

En caso de usar una placa comercial **el montaje es realmente sencillo**. En primer lugar alimentamos la electrónica del módulo Vcc y GND a Vcc y GND de Arduino mediante los terminales existentes.

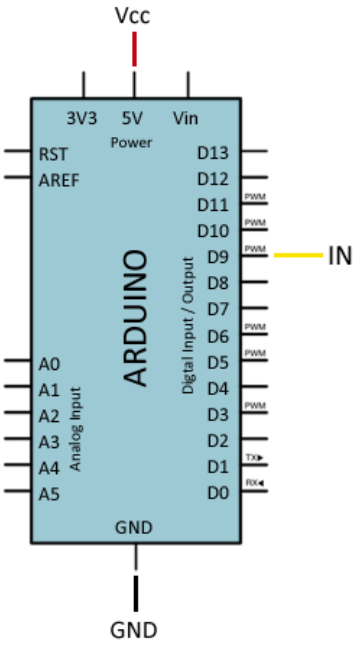
Por otro lado conectamos la carga a la clema de tres conexiones. Siempre debemos conectar uno de los polos de la carga al terminal C, que habitualmente es el terminal del medio.

El otro polo de la carga lo conectaremos al terminal NO o NC, dependiendo de si cuando el relé este desactivado queremos que el secundario este abierto (NO), o cerrado (NC).

Finalmente conectamos el pin de señal a una salida digital de Arduino. Si empleamos una placa con varios canales, conectaríamos cada uno de los canales directamente a una salida digital.

Carga: Tensión máxima 250VAC.
Intensidad máxima 10A.

La conexión, vista desde Arduino, sería la siguiente.



El código necesario es muy sencillo, y solo necesitamos tratarla como cualquier otra salida digital tal y como vimos anteriormente en la sección de salidas digitales.

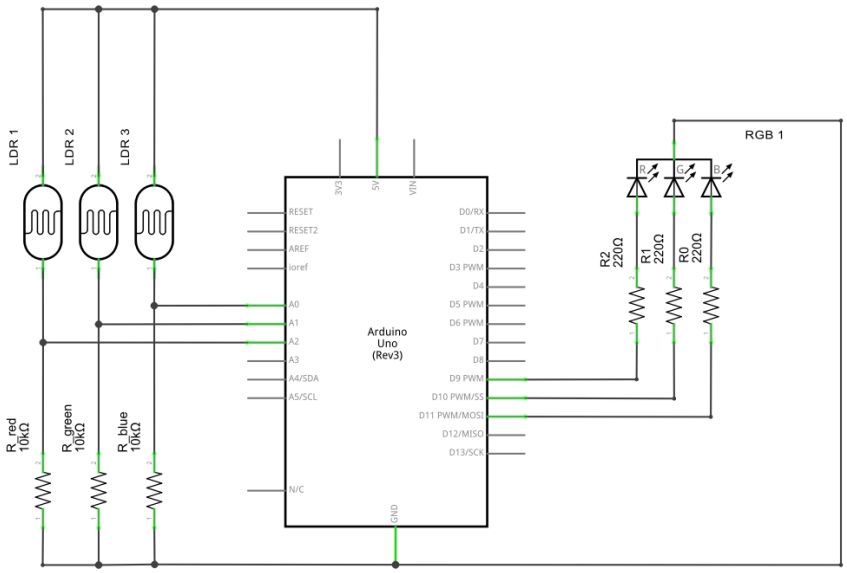
Por ejemplo, el siguiente código simplemente enciende y apaga la carga en intervalos de 10 segundos.

```
// DECLARACIÓN DE VARIABLES
const int pin = 9;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  Serial.begin(9600);     //iniciar puerto serie
  pinMode(pin, OUTPUT);   //definir pin como salida
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
  digitalWrite(pin, HIGH);    // poner el Pin en HIGH
  delay(10000);                // esperar un segundo
  digitalWrite(pin, LOW);     // poner el Pin en LOW
  delay(10000);                // esperar un segundo
}
```

LED RGB CON SENSOR DE LUZ LDR

Esta práctica será un poco más compleja que las anteriores al unificar varios componentes y emplear los pines con modulación PWM. Vamos a utilizar por pimera vez el LED RGB y los sensores de luz.

Un LED con cuatro pins es un diodo LED RGB de cátodo común. Este diodo LED tiene los elementos rojo, verde y azul separados en su interior, y un masa común (el cátodo). Al aparecer una diferencia de tensión entre el cátodo y las tensiones de salida de los pins de Arduino en formato PWM (los cuales se conectan a los ánodos del LED a través de unas resistencias de 220 ohmios), será posible hacer que el LED varíe la iluminación de sus tres colores. Observar que el pin más largo del LED se conecta a masa. Conectar los otros tres pins del LED a los pins digitales 9,10 y 11 de Arduino y en serie con cada uno de ellos una resistencia de 220 ohmios. Asegurarse que cada cable del LED se conecta correctamente a su pin PWM correspondiente.



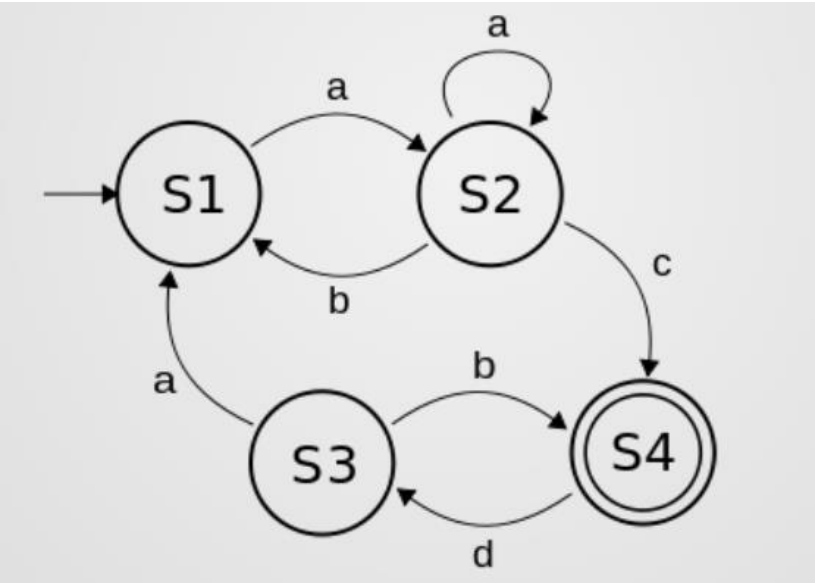

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int PinLedVerde = 9;
const int PinLedAzul = 10;
const int PinLedRojo = 11;

const int PinEntradaLDR_Rojo = A0;
const int PinEntradaLDR_Verde = A1;
```

```
const int PinEntradaLDR_Azul = A2;

int ValorRojo = 0;
int ValorVerde = 0;
int ValorAzul = 0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
    Serial.begin(9600);                // inicializamos la comunicación serie
    pinMode(PinLedVerde, OUTPUT);
    pinMode(PinLedAzul, OUTPUT);
    pinMode(PinLedRojo, OUTPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
    ValorRojo = analogRead(PinEntradaLDR_Rojo)/4;
    delay(5);
    ValorVerde = analogRead(PinEntradaLDR_Verde)/4;
    delay(5);
    ValorAzul = analogRead(PinEntradaLDR_Azul)/4;

    Serial.print("Mapa de color");
    Serial.print(ValorRojo);
    Serial.print(ValorVerde);
    Serial.println(ValorAzul);

    analogWrite(PinLedRojo, ValorRojo);
    analogWrite(PinLedVerde, ValorVerde);
    analogWrite(PinLedAzul, ValorAzul);
}
```

MAQUINA DE ESTADOS

La máquina de estados es unos de los conceptos más importantes en la programación de automátas. Un "estado" podría definirse como una funcionalidad que está realizando el autómata hasta que una señal o variable le haga cambiar de estado.

La mejor forma de clarificar este concepto será con un ejemplo.



En este caso, vamos a realizar la primera de ellas. Utilizar un pulsador como accionador de cambio de estado.

```
// DECLARACIÓN DE VARIABLES
const int buttonPin = 2;     // Pin del pulsador
const int ledPin =  13;      // Pin del led

int buttonState = 0;         // Estado del pulsador

/* Nuestra maquina de estados funciona de la siguiente manera:
 *  0 : Apagado
 *  1 : Encendido
 *  2 : Parpadeo
 */
int estado = 0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
  // read the state of the pushbutton value:
```

```
  buttonState = digitalRead(buttonPin);
if (buttonState == HIGH) {
  estado ++;
  delay(500);
  if (estado==3){
  estado=0;
  }
}
switch (estado){
    case 0:    // Estado 0 apagado
        digitalWrite(ledPin, LOW);
      break;
    case 1:    // Estado 1 encendido
      digitalWrite(ledPin, HIGH);
      break;
    case 2:    // Estado 2 parpadeando
      digitalWrite(ledPin, LOW);
      delay(500);
      digitalWrite(ledPin, HIGH);
      delay(500);
      break;
  }
}
```

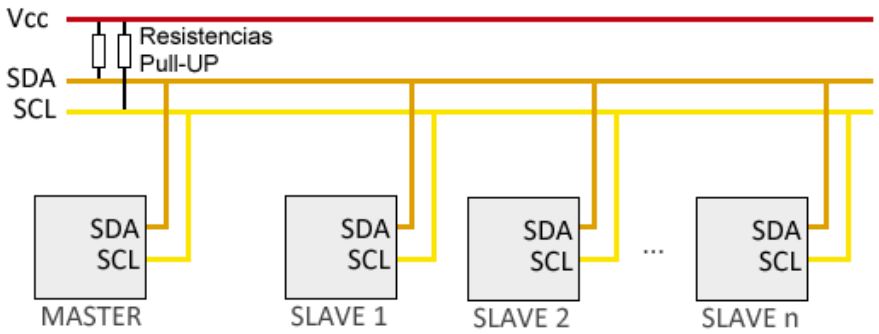
PRÁCTICAS INTERMEDIAS

PANTALLA 1602 CON I2C PARA MOSTRAR LOS DATOS

El BUS I2C

El estándar I2C (Inter-Integrated Circuit) fue desarrollado por Philips en 1982 para la comunicación interna de dispositivos electrónicos en sus artículos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.

El bus I2C **requiere únicamente dos cables** para su funcionamiento, uno para la señal de reloj (CLK) y otro para el envío de datos (SDA).



En el bus **Cada dispositivo dispone de una dirección**, que se emplea para acceder a los dispositivo de forma individual. Esta dirección puede ser fijada por hardware (en cuyo caso, frecuentemente, se pueden modificar los últimos 3 bits mediante jumpers o interruptores) o totalmente por software.

Arduino dispone de **soporte I2C por hardware** vinculado físicamente a ciertos pines. También es posible emplear cualquier otro grupo de pines como bus I2C a través de sofware, pero en ese caso la velocidad será mucho menor.

En un mundo ideal sabríamos la dirección del dispositivo que compramos. Pero en algunas ocasiones, **el fabricante no nos facilita la dirección del dispositivo** o incluso lo proporciona de forma incorrecta.

Esta es una circunstancia común y nada preocupante. Para eso disponemos de un sketch llamado "**Scanner I2C**" que realiza un barrido por todas las posibles direcciones del bus, y muestra el resultado en caso de encontrar un dispositivo en la dirección.

Con el siguiente programa de scanner_I2C podemos encontrar cómodamente la dirección de nuestro dispositivo I2C:

```
#include <Wire.h>

void setup()
{
  Wire.begin();

  Serial.begin(9600);
  while (!Serial);             // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
```

```
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

Descargamos e instalamos la librería LiquidCrystal-I2C que se obtiene aquí:

https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

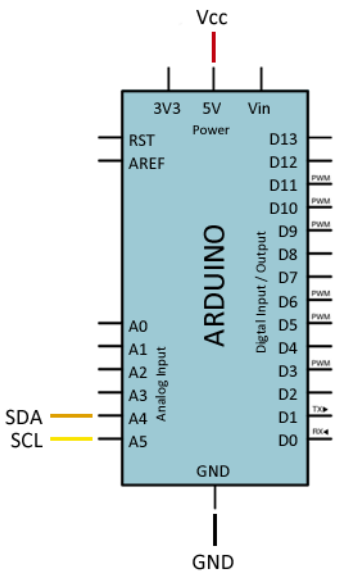
Para instalar una librería nueva abrimos el IDE de arduino y vamos a:

Programa -> incluir librería -> Añadir librería .ZIP

ESQUEMA MONTAJE

La conexión es sencilla, simplemente alimentamos el módulo desde Arduino mediante GND y 5V y conectamos el pin SDA y SCL de Arduino con los pines correspondientes del controlador LCD I2C.

Mientras que la conexión vista desde el lado de Arduino quedaría así.



Una vez que tengamos la dirección de nuestro dispositivo I2C probamos el primer programa de ejemplo:

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// DECLARACIÓN DE VARIABLES
LiquidCrystal_I2C lcd(0x3F, 16, 2);  // Inicia el LCD en la dirección 0x3F, con 16
caracteres y 2 líneas

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
lcd.begin();                      // inicializamos el objeto lcd
lcd.setCursor(0, 0);             // Colocamos el cursor para empezar a escribir en 0,0
lcd.print("Linea 1");
lcd.setCursor(0, 1);
lcd.print("Linea 2");
delay(2500);
lcd.clear();                    // Limpiamos el LCD
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
}
```

Posibles prácticas recomendadas:

1. Hacer un contador de horas, minutos y segundos en el display
2. Enviar un texto por puerto serie y mostrarlo en la pantalla
3. Intercambiar mensajes mostrador en el display con un botón de entrada

SONIDOS CON EL BUZZER ACTIVO Y NOTAS CON EL PASIVO

Los buzzers son dispositivos que permiten convertir una señal eléctrica en una onda de sonido. Un buzzer activo incorpora un oscilador simple por lo que **únicamente es necesario suministrar corriente al dispositivo** para que emita sonido. Por el contrario, los buzzer pasivos necesitan recibir una onda a una frecuencia determinando pudiendo de ese modo **variar el tono emitido** que emite al altavoz, lo que nos permite generar melodías.

La conectividad de este componente es muy sencilla. Solo dos pines que no requieren de resistencia en serie. Se utiliza para bajar la intensidad de sonido que reproduce el buzzer.

Cuando se trata de **Buzzer pasivos** introducimos dos funciones nuevas muy sencillas de utilizar, **tone(Pin, Frecuencia, duración)** y la funcion **noTone(pin)** que se utiliza cuando se desea reproducir tonos difernetes en varios pines.

Marcha Imperial Buzzer

```
// NB: ALL NOTES DEFINED WITH STANDARD ENGLISH NAMES, EXCEPT FROM "A"
//THAT IS CALLED WITH THE ITALIAN NAME "LA" BECAUSE A0,A1...ARE THE ANALOG PINS ON
ARDUINO.
// (Ab IS CALLED Ab AND NOT LAb)
#define  C0 16.35
#define Db0 17.32
#define D0  18.35
#define Eb0 19.45
#define E0  20.60
#define F0  21.83
#define Gb0 23.12
#define G0  24.50
#define Ab0 25.96
#define LA0 27.50
#define Bb0 29.14
#define B0  30.87
#define C1  32.70
#define Db1 34.65
#define D1  36.71
#define Eb1 38.89
#define E1  41.20
#define F1  43.65
#define Gb1 46.25
#define G1  49.00
#define Ab1 51.91
#define LA1 55.00
```

```
#define Bb1 58.27
#define B1  61.74
#define C2  65.41
#define Db2 69.30
#define D2  73.42
#define Eb2 77.78
#define E2  82.41
#define F2  87.31
#define Gb2 92.50
#define G2  98.00
#define Ab2 103.83
#define LA2 110.00
#define Bb2 116.54
#define B2  123.47
#define C3  130.81
#define Db3 138.59
#define D3  146.83
#define Eb3 155.56
#define E3  164.81
#define F3  174.61
#define Gb3 185.00
#define G3  196.00
#define Ab3 207.65
#define LA3 220.00
#define Bb3 233.08
#define B3  246.94
#define C4  261.63
#define Db4 277.18
#define D4  293.66
#define Eb4 311.13
#define E4  329.63
#define F4  349.23
#define Gb4 369.99
#define G4  392.00
#define Ab4 415.30
#define LA4 440.00
#define Bb4 466.16
#define B4  493.88
#define C5  523.25
#define Db5 554.37
#define D5  587.33
#define Eb5 622.25
#define E5  659.26
#define F5  698.46
#define Gb5 739.99
#define G5  783.99
#define Ab5 830.61
#define LA5 880.00
#define Bb5 932.33
#define B5  987.77
#define C6  1046.50
#define Db6 1108.73
#define D6  1174.66
#define Eb6 1244.51
#define E6  1318.51
#define F6  1396.91
```

```
#define Gb6 1479.98
#define G6  1567.98
#define Ab6 1661.22
#define LA6 1760.00
#define Bb6 1864.66
#define B6  1975.53
#define C7  2093.00
#define Db7 2217.46
#define D7  2349.32
#define Eb7 2489.02
#define E7  2637.02
#define F7  2793.83
#define Gb7 2959.96
#define G7  3135.96
#define Ab7 3322.44
#define LA7 3520.01
#define Bb7 3729.31
#define B7  3951.07
#define C8  4186.01
#define Db8 4434.92
#define D8  4698.64
#define Eb8 4978.03
// DURATION OF THE NOTES
#define BPM 120    //  you can change this value changing all the others
#define H 2*Q //half 2/4
#define Q 60000/BPM //quarter 1/4
#define E Q/2   //eighth 1/8
#define S Q/4 // sixteenth 1/16
#define W 4*Q // whole 4/4

void setup() {
pinMode(8, OUTPUT);

}

// the loop routine runs over and over again forever:
void loop() {
  //tone(pin, note, duration)
    tone(8,LA3,Q);
    delay(1+Q); //delay duration should always be 1 ms more than the note in order to
separate them.
    tone(8,LA3,Q);
    delay(1+Q);
    tone(8,LA3,Q);
    delay(1+Q);
    tone(8,F3,E+S);
    delay(1+E+S);
    tone(8,C4,S);
    delay(1+S);

    tone(8,LA3,Q);
    delay(1+Q);
    tone(8,F3,E+S);
    delay(1+E+S);
    tone(8,C4,S);
    delay(1+S);
```

```
tone(8,LA3,H);
delay(1+H);

tone(8,E4,Q);
delay(1+Q);
tone(8,E4,Q);
delay(1+Q);
tone(8,E4,Q);
delay(1+Q);
tone(8,F4,E+S);
delay(1+E+S);
tone(8,C4,S);
delay(1+S);

tone(8,Ab3,Q);
delay(1+Q);
tone(8,F3,E+S);
delay(1+E+S);
tone(8,C4,S);
delay(1+S);
tone(8,LA3,H);
delay(1+H);

tone(8,LA4,Q);
delay(1+Q);
tone(8,LA3,E+S);
delay(1+E+S);
tone(8,LA3,S);
delay(1+S);
tone(8,LA4,Q);
delay(1+Q);
tone(8,Ab4,E+S);
delay(1+E+S);
tone(8,G4,S);
delay(1+S);

tone(8,Gb4,S);
delay(1+S);
tone(8,E4,S);
delay(1+S);
tone(8,F4,E);
delay(1+E);
delay(1+E);//PAUSE
tone(8,Bb3,E);
delay(1+E);
tone(8,Eb4,Q);
delay(1+Q);
tone(8,D4,E+S);
delay(1+E+S);
tone(8,Db4,S);
delay(1+S);

tone(8,C4,S);
delay(1+S);
tone(8,B3,S);
delay(1+S);
```

```
tone(8,C4,E);
delay(1+E);
delay(1+E);//PAUSE QUASI FINE RIGA
tone(8,F3,E);
delay(1+E);
tone(8,Ab3,Q);
delay(1+Q);
tone(8,F3,E+S);
delay(1+E+S);
tone(8,LA3,S);
delay(1+S);

tone(8,C4,Q);
delay(1+Q);
 tone(8,LA3,E+S);
delay(1+E+S);
tone(8,C4,S);
delay(1+S);
tone(8,E4,H);
delay(1+H);

 tone(8,LA4,Q);
delay(1+Q);
tone(8,LA3,E+S);
delay(1+E+S);
tone(8,LA3,S);
delay(1+S);
tone(8,LA4,Q);
delay(1+Q);
tone(8,Ab4,E+S);
delay(1+E+S);
tone(8,G4,S);
delay(1+S);

tone(8,Gb4,S);
delay(1+S);
tone(8,E4,S);
delay(1+S);
tone(8,F4,E);
delay(1+E);
delay(1+E);//PAUSE
tone(8,Bb3,E);
delay(1+E);
tone(8,Eb4,Q);
delay(1+Q);
tone(8,D4,E+S);
delay(1+E+S);
tone(8,Db4,S);
delay(1+S);

tone(8,C4,S);
delay(1+S);
tone(8,B3,S);
delay(1+S);
tone(8,C4,E);
delay(1+E);
```

```
    delay(1+E);//PAUSE QUASI FINE RIGA
    tone(8,F3,E);
    delay(1+E);
    tone(8,Ab3,Q);
    delay(1+Q);
    tone(8,F3,E+S);
    delay(1+E+S);
    tone(8,C4,S);
    delay(1+S);

    tone(8,LA3,Q);
    delay(1+Q);
     tone(8,F3,E+S);
    delay(1+E+S);
    tone(8,C4,S);
    delay(1+S);
    tone(8,LA3,H);
    delay(1+H);

    delay(2*H);

}
```

El código necesario para utilizar los buzzer activos es igual que el que empleamos en una salida digital, tal y como vimos al principio.

Por ejemplo, el siguiente código simplemente encendería el buzzer activo durante 5 segundos, dejándolo descansar 20 segundos.

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int pin_buzzer_activo = 9;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  pinMode(pin_buzzer_activo, OUTPUT);  //definir pin como salida
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  digitalWrite(pin_buzzer_activo, HIGH);   // poner el Pin en HIGH
  delay(5000);               // esperar 5 segundos
  digitalWrite(pin_buzzer_activo, LOW);    // poner el Pin en LOW
  delay(20000);               // esperar 20 segundos

}
```

LEER TEMPERATURA Y HUMEDAD DHT-11

Estos sensores disponen de un procesador interno que realiza el proceso de medición, proporcionando la medición mediante una señal digital, por lo que **resulta muy sencillo obtener la medición desde un microprocesador como Arduino**.

Las caracterísitcas de este tipo de sensores aunque se quedan cortas para usos comerciales son perfectos para pruebas, formación o proyectos que no requieran una gran precisión:

- Medición de temperatura entre 0 a 50, con una precisión de 2ºC
- Medición de humedad entre 20 a 80%, con precisión del 5%.
- Frecuencia de muestreo de 1 muestras por segundo (1 Hz)

La conexión del DH11 y el DHT22 son idénticas, ya que como hemos comentado la única diferencia entres modelos son sus prestaciones. En ambos casos, disponemos de 4 patillas, de las cuales usaremos 3, Vcc, Output y GND.

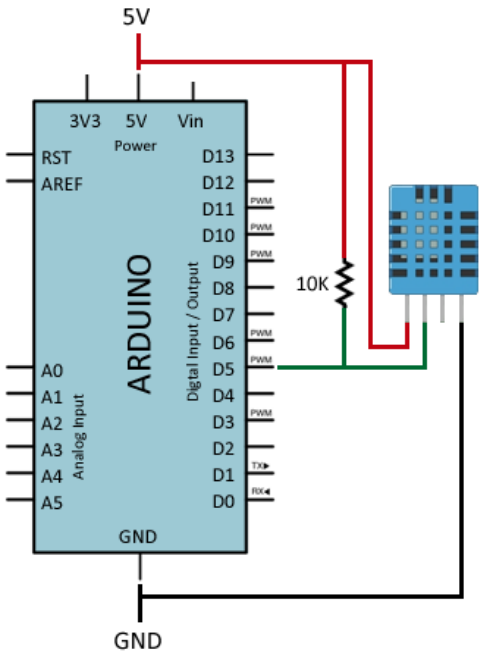
La conexión del módulo DH-11, es muy sencilla. Disponemos de 3 patillas: Señal, Vcc y GND.

Aprovechamos esta práctica para comentar otro modelo de sensor de mejores prestaciones pero con un encapsulado similar. El DHT-22 tiene:

- Medición de temperatura entre -40 a 125, con una precisión de 0.5ºC
- Medición de humedad entre 0 a 100%, con precisión del 2-5%.
- Frecuencia de muestreo de 2 muestras por segundo (0.5 Hz)

Conectar el sensor DHT-11 o DHT-22 sin su formato en módulo es sencillo, simplemente alimentamos desde Arduino al sensor a través de los pines GND y Vcc del mismo. Por otro lado, conectamos la salida Output a una entrada digital de Arduino. Necesitaremos poner una resistencia de 10K entre Vcc y el Pin Output.

El esquema eléctrico queda como la siguiente imagen



Cualquiera de los sensores DHT11 y DHT22 usan su propio sistema de comunicación bidireccional mediante un único conductor, empleando señales temporizadas.

En cada envío de medición el sensor envía un total de 40bits, en 4ms. Estos 40 bits corresponden con 2 Bytes para la medición de humedad, 2 Bytes para la medición de temperatura, más un Byte final para la comprobación de errores (8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum)

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR
#include <DHT.h>

// Descomentar el Sensor que vayamos a utilizar
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// DECLARACIÓN DE VARIABLES
const int DHTPin = 5;
DHT dht(DHTPin, DHTTYPE);
float h = 0;
float t = 0;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);            // inicializamos la comunicación Serie
  dht.begin();                   // inicializamos el objeto dht

}
```

```
// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{

  // Leer la temperatura y la humedad toma al rededor de 250milisegundos
  h = dht.readHumidity();
  t = dht.readTemperature();
  delay(250);

  if (isnan(h) || isnan(t)) {

    Serial.println("Error de lectura");
  }
  else{
  //mostrar valores en el LCD 1602
  Serial.print("Humedad: ");
  Serial.println(h);
  Serial.print("Temp: ");
  Serial.println(t);
  }
}
```

LEER DISTANCIA CON EL SENSOR ULTRASONIDOS

Los sensores de ultrasonidos **son sensores de baja precisión**. La orientación de la superficie a medir puede provocar que la onda se refleje, falseando la medición. Además, no resultan adecuados en entornos con gran número de objetos, dado que el sonido rebota en las superficies generando ecos y falsas mediciones. Tampoco son los más apropiados para el funcionamiento en el exterior y al aire libre.

Pese a esta baja precisión, que impide conocer con precisión la distancia a un objeto, **los sensores de ultrasonidos son ampliamente empleados**. En robótica es habitual montar uno o varios de estos sensiores, por ejemplo, para detección de obstáculos, determinar la posición del robot, crear mapas de entorno, o resolver laberintos.

Teoricamente el rango de medida de los sensores HC-SR04 es de 2cm a 400 cm, con una resolución de 0.3cm. En la práctica, sin embargo, el rango de medición real es mucho más limitado, en torno a 10cm a 2 metros.

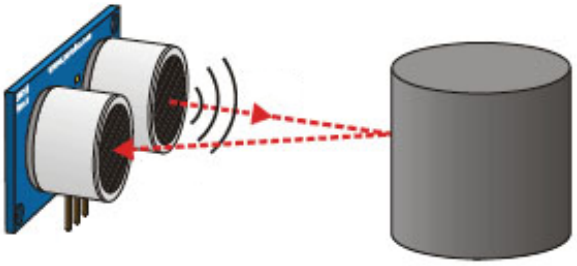
El sensor se basa simplemente en **medir el tiempo entre el envío y la recepción de un pulso sonoro**. Sabemos que la velocidad del sonido es 343 m/s en condiciones de temperatura 20 ºC, 50% de humedad, presión atmosférica a nivel del mar. Transformando unidades resulta

$$343\frac{m}{s} \cdot 100\frac{cm}{m} \cdot \frac{1}{1000000}\frac{s}{\mu s} = \frac{1}{29.2}\frac{cm}{\mu s}$$

Es decir, el sonido tarda 29,2 microsegundos en recorrer un centímetro. Por tanto, podemos obtener la distancia a partir del tiempo entre la emisión y recepción del pulso mediante la siguiente ecuación.

$$Distancia(cm) = \frac{Tiempo(\mu s)}{29.2 \cdot 2}$$

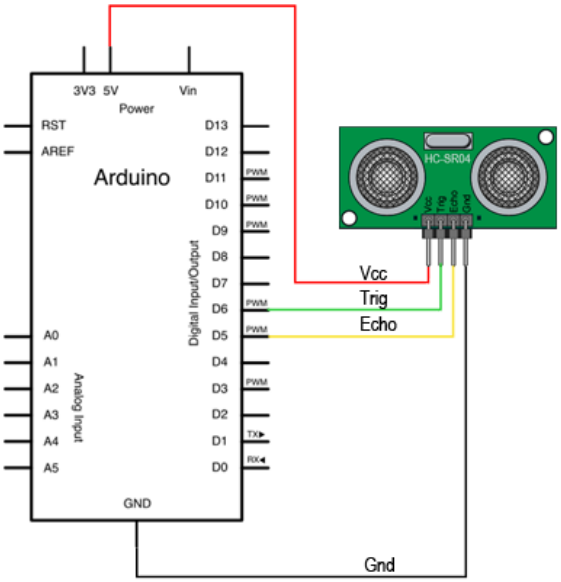
El motivo de dividir entre dos el tiempo es porque hemos medido el tiempo que tarda el pulso en ir y volver, por lo que la distancia recorrida por el pulso es el doble de la que queremos medir.



Tiempo = 2 * (Distancia / Velocidad)

Distancia = Tiempo · Velocidad / 2

El esquema eléctrico que necesitamos es el siguiente.



Para activar el sensor necesitamos generar un pulso eléctrico en el pin **Trigger** (disparador) de al menos 10us. Previamente, pondremos el pin a Low durante 4us para asegurar un disparo limpio.

Posteriormente usamos la función "**pulseIn**" para obtener el tiempo requerido por el pulso para volver al sensor. Finalmente, convertimos el tiempo en distancia mediante la ecuación correspondiente.

Observar que intentamos emplear siempre aritmética de enteros, **evitando usar números en coma flotante**. Esto es debido a que las operaciones en coma flotante ralentizan mucho el procesador, y suponen cargar un gran número de librerías en memoria.

```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR

// DECLARACIÓN DE VARIABLES
const int EchoPin = 5;
const int TriggerPin = 6;
const int LedPin = 13;

// DECLARACIÓN DE FUNCIONES
int ping(int TriggerPin, int EchoPin) {
  long duration, distanceCm;

  digitalWrite(TriggerPin, LOW);  //para generar un pulso limpio ponemos a LOW 4us
  delayMicroseconds(4);
  digitalWrite(TriggerPin, HIGH);  //generamos Trigger (disparo) de 10us
  delayMicroseconds(10);
  digitalWrite(TriggerPin, LOW);
  duration = pulseIn(EchoPin, HIGH);  //medimos el tiempo entre pulsos, en
microsegundos
  distanceCm = duration * 10 / 292/ 2;   //convertimos a distancia, en cm
     return distanceCm;
}

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);                // inicializamos la comunicación Serie
  pinMode(LedPin, OUTPUT);
  pinMode(TriggerPin, OUTPUT);
  pinMode(EchoPin, INPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  int cm = ping(TriggerPin, EchoPin);
  Serial.print("Distancia: ");
  Serial.println(cm);
  delay(1000);
}
```

USO DE SERVOMOTORES CON POTENCIOMETRO

Habitualmente los servomotores disponen de **un rango de movimiento de entre 0 a 180º.** Es decir, no son capaces de dar la vuelta por completo. Internamente un servo consta de un mecanismo reductor. Por tanto **proporcionan un alto par y un alto grado de precisión** (incluso décimas de grado). Por contra, las **velocidades de giro son pequeñas** frente a los motores de corriente continua.

El servomotor SG-90 es un pequeño servo, ligero, y barato, que dispone de engranajes de plástico. Muchos dispositivos, como torretas y partes de robots, están diseñados para utilizar servos de este tamaño.

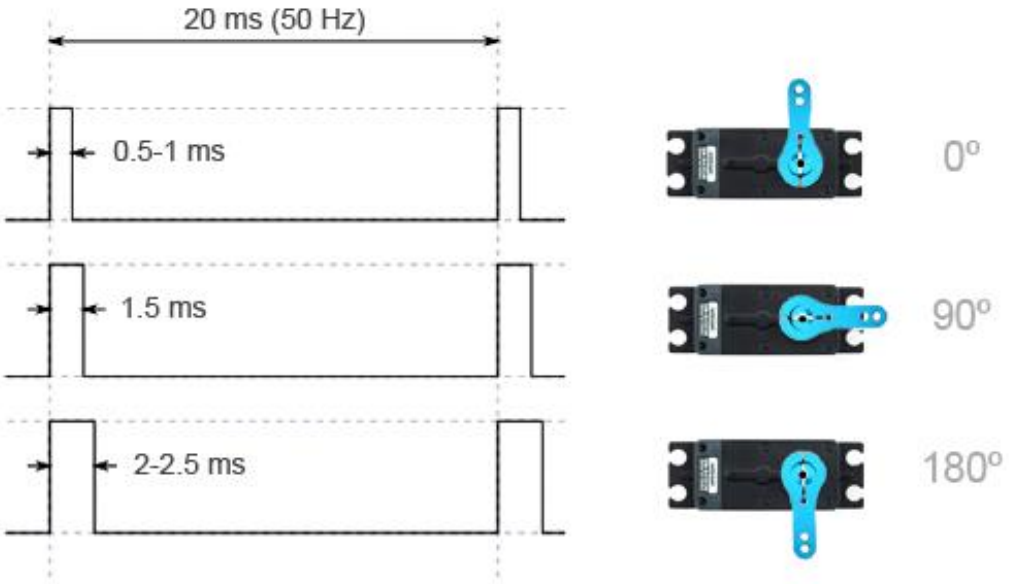
- Torque: 1.4 kg·cm
- Velocidad: 0.1 seg/60º (4.8V) y 0.08 seg/60º (6V)
- Dimensiones: 21.5 x 11.8 x 22.7mm
- Peso: 9g

FUNCIONAMIENTO DEL SERVOMOTOR

Internamente un servo está constituido por un motor de corriente continua, acoplado a un reductor para reducir la velocidad de giro, junto con la electrónica necesaria para controlar su posición.

Frecuentemente, se dispone de un potenciómetro unido al eje del servo, que permite al servo para conocer la posición del eje. Esta información es tratada por un controlador integrado que se encarga de actuar sobre el motor para alcanzar la posición deseada.

La comunicación de la posición deseada se realiza mediante la transmisión de una señal pulsada con periodo de 20ms. El ancho del pulso determina la posición del servo.



La relación entre el ancho del pulso y el ángulo depende del modelo del motor. Por ejemplo, algunos modelos responden con 0º a un pulso de 500 ms, y otros a un pulso de 1000 ms

En general, en todos los modelos:

- Un pulso entre 500-1000 us corresponde con 0º
- Un pulso de 1500 ms corresponde con 90º (punto neutro)
- Un pulso entre 2000-2500us corresponde con 180º

Por tanto, variando la señal en microsegundos podemos disponer de una precisión teórica de 0.18-0.36º, siempre que la mecánica del servo acompañe.

CONECTAR EL SERVOMOTOR

Conectar un servo a Arduino es sencillo. El servo dispone de tres cables, dos de alimentación (GND y Vcc) y uno de señal (Sig).

El color de estos cables suele tener dos combinaciones:

– Marrón (GND), Rojo (Vcc) y Naranja (Sig)

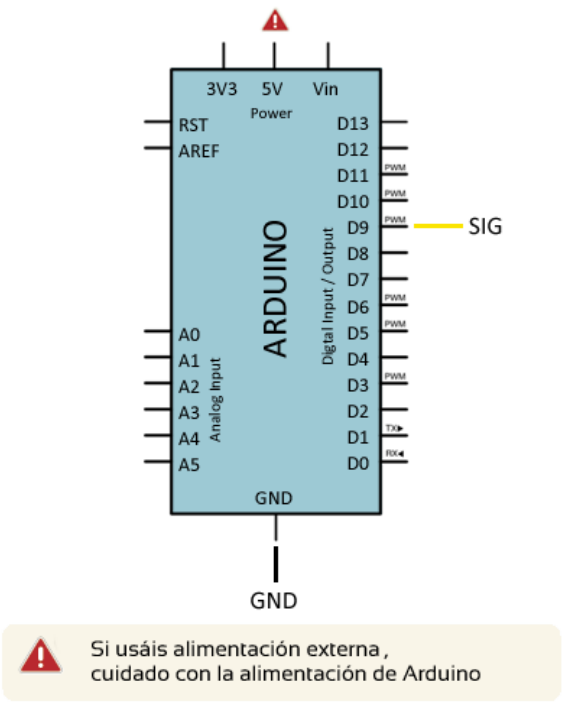
– Negro (GND), Rojo (Vcc) y Blanco (Sig)

En general, **la alimentación a los servos se realizará desde una fuente de tensión externa** (una batería o fuente de alimentación) a una tensión de 5V-6.5V, siendo 6V la tensión idónea.

Arduino puede llegar a proporcionar corriente suficiente para encender un servo pequeño (SG90), suficiente para hacer unos cuantos proyectos de prueba.

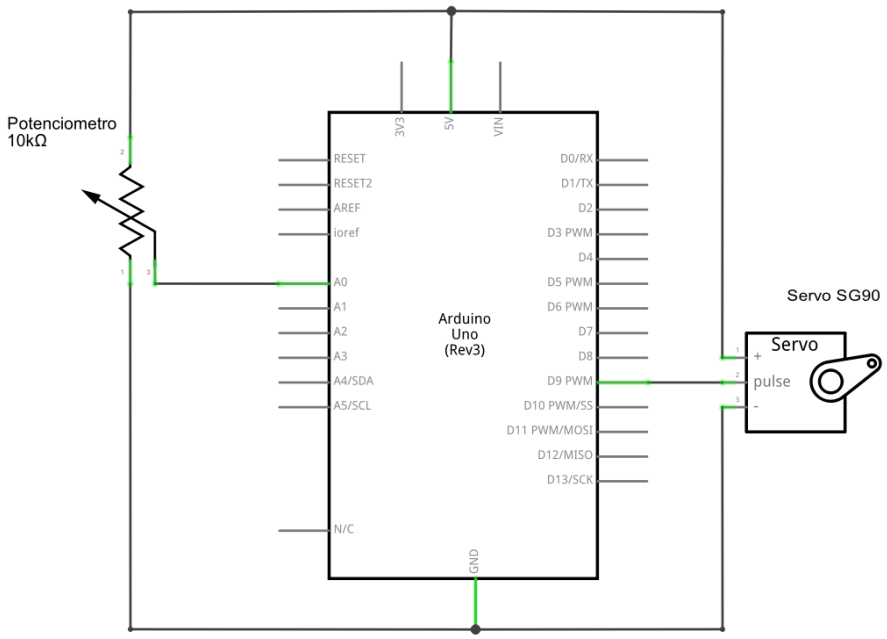
Sin embargo no dispone de corriente suficiente para actuar un servo grande (MG996R) o incluso varios servos pequeños. Una sobrecarga en la capacidad de corriente suministrada por el Arduino provocará su reinicio.

Al usar varias fuente de tensión recordar poner siempre en común todos los GND. De lo contrario podríais dañar algún componente.



El control de servomotor en Arduino es muy sencillo, ya que el IDE Standard proporciona la librería "servo.h", que permite controlar simultáneamente hasta 12 servos en Arduino Uno/Nano y hasta 48 servos en Arduino Mega.

La práctica que vamos a realizar será la de mapear la posición del servomotor con un potenciómetro.



```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR
#include <Servo.h>

// DECLARACIÓN DE VARIABLES
Servo MiServo;
const int PinPot = A0;
const int PinServo = 9;
int ValorPot;
int ValorMinPot = 1023;
int ValorMaxPot = 0;
int angulo;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);                // inicializamos la comunicación serie
  MiServo.attach(PinServo);          // inicializamos el objeto Servomotor

  while(millis()<10000){            // Inicializamos los márgenes max y min del
potenciómetro. Muy útil para sensores LDR, NTC, sensores de llama…
    ValorPot = analogRead(PinPot);
    if (ValorPot > ValorMaxPot){
      ValorMaxPot = ValorPot;
    }
    if (ValorPot < ValorMinPot){
      ValorMinPot = ValorPot;
    }
      Serial.print("max: ");
      Serial.print(ValorMaxPot);
      Serial.print(" |  min: ");
      Serial.println(ValorMinPot);
    }
    Serial.println("calibrado!");
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  ValorPot = analogRead(PinPot);
  angulo = map(ValorPot, ValorMinPot, ValorMaxPot, 0, 180);
  Serial.print("Potenc: ");
  Serial.println(ValorPot);
  Serial.print("Angulo: ");
  Serial.println(angulo);
  Serial.println("--------------------------");
  MiServo.write(angulo);    // movemos el servomotor al angulo mapeado
  delay(500);
}
```

USO DE MOTORES PASO A PASO

El 28BYJ-48 tiene un paso de 5.625 grados (64 pasos por vuelta). El reductor interno tiene una relación de 1/64. Combinados, la precisión total es de 4096 pasos por vuelta, equivalente a un paso de 0.088º, que es una precisión muy elevada.

El tipo de motor PAP tiene un par máximo tras el reductor de 3N•cm (0.3Kgf•cm). Así como una frecuencia máxima es de 100Hz, lo que supone unos 40 segundos por vuelta, o equivalentemente una velocidad de giro máxima en torno a 1.5 rpm.

El 28BYJ-48 es un motor ampliamente utilizado en aplicaciones industriales, como el control de válvulas hidráulicas y neumáticas. En el campo de los hobbies se usa para hacer pequeños robots articulados, brazos robóticos, posicionar sensores, o girar la mesa de un escanner.

FUNCIONAMIENTO DEL MOTOR PASO A PASO

Para el control del 28BYJ-48 vamos a usar una placa con el integrado ULN2003. El ULN2003 es una agrupación de 7 Par Darlington de los cuales usaremos 4, uno por cada fase del motor paso a paso.

Un par Darlington es un dispositivo que nos permite suministrar una corriente superior a las que podríamos usando las salidas de Arduino directamente.

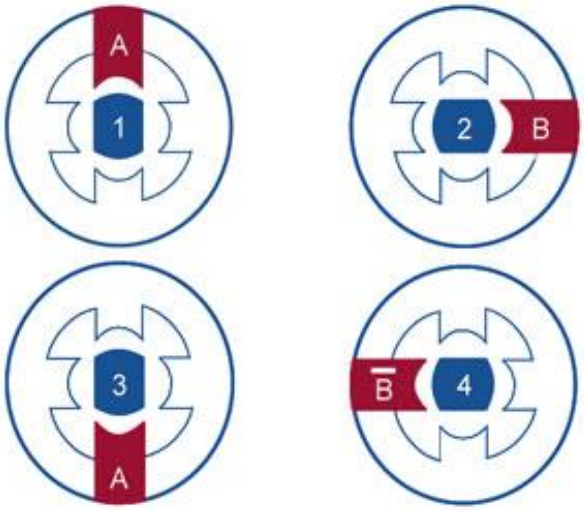
El 28BYJ-48 es un motor bipolar, por lo que no necesitamos invertir la corriente que atraviesa las bobinas para hacerlo funcionar. Por tanto, no necesitamos un controlador complicado como el A4988 o el DRV8825, ni siquiera un simple puente-H como el L298N.

Aplicando un control "todo o nada", es decir, encendiendo por completo o apagando por completo una bobina, únicamente tenemos que activar las bobinas en la secuencia correcta.

Existen varias secuencias posibles, y acontinuación vamos a ver las tres más empleadas.

SECUENCIA 1-FASE

En secuencia de 1-fase encendemos una única bobina cada vez.

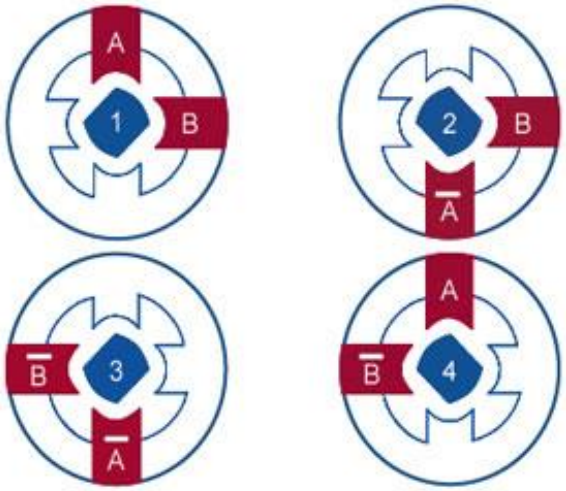


Llevando esta secuencia de encendido a una tabla, que posteriormente usaremos para el código, la secuencia quedaría de la siguiente forma,

Paso	A	B	A'	B'
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON

SECUENCIA 2-FASES

En la secuencia en 2-fases encendemos dos bobinas correlativas en cada fase. Al hacer trabajar dos bobinas simultáneamente en cada paso el campo magnético generado es superior (un 41% más) por lo que el motor tiene más par y, en general, presenta un mejor comportamiento. Como punto negativo, aumentamos el consumo energético al doble.



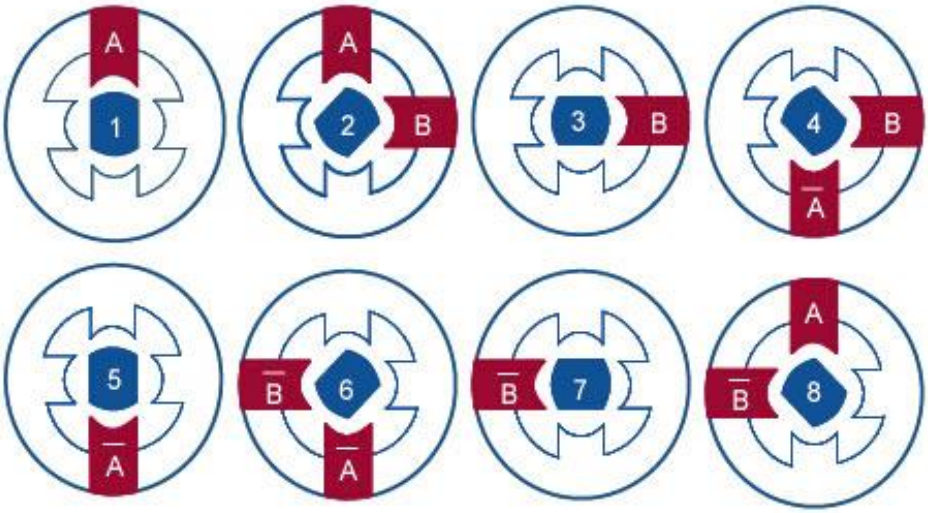
Que expresado en forma de tabla resulta,

Paso	A	B	A'	B'
1	ON	OFF	OFF	ON
2	ON	ON	OFF	OFF
3	OFF	ON	ON	OFF
4	OFF	OFF	ON	ON

SECUENCIA MEDIO PASOS

La última secuencia que vamos a ver es la secuencia en medio paso (half-step). Aquí encendemos alternativamente uno y dos bobinas.

Con esta secuencia conseguimos una precisión de la mitad del paso. El par desarrollado varía ya que en algunos pasos activamos dos bobinas y en otras solo una, pero a la vez el giro se encuentra más "guiado", por lo que en general ambos efectos se compensan y el funcionamiento es bueno, salvo en aplicaciones donde estemos muy al límite del par máximo.

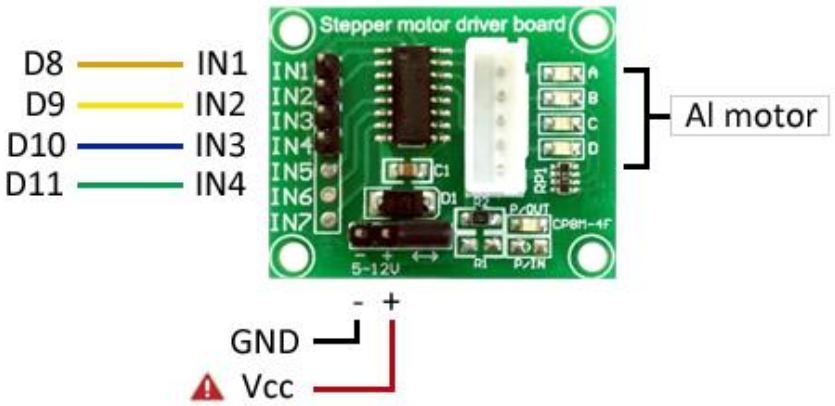


Expresando la secuencia en forma de tabla resulta

Medio-paso	A	B	A'	B'
1	ON	OFF	OFF	OFF
2	ON	ON	OFF	OFF
3	OFF	ON	OFF	OFF
4	OFF	ON	ON	OFF
5	OFF	OFF	ON	OFF
6	OFF	OFF	ON	ON
7	OFF	OFF	OFF	ON
8	ON	OFF	OFF	ON

CONECTAR EL MOTOR PASO A PASO

La conexión del 28BYJ-48 a Arduino a través del módulo ULN2003 es bastante sencilla.



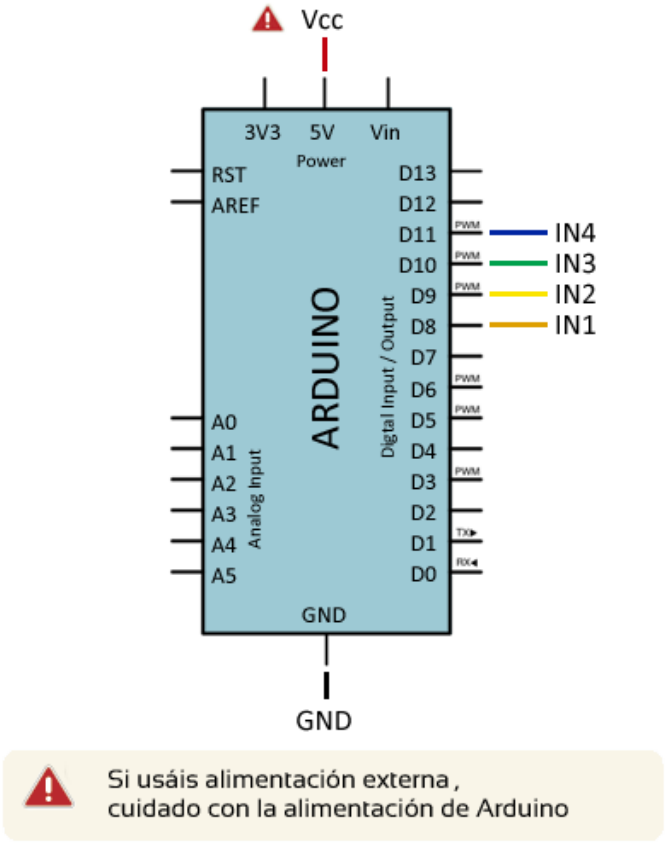
Vcc entre 5 y 12V. Al usar alimentación externa SIEMPRE poner con GND común.

En primer lugar, conectamos el 28BYJ-48 a la placa con el integrado ULN2003 con la clema de conexión incorporada. Los terminales solo encajan en una posición, por lo que no hay riesgo de conectarlo incorrectamente.

Por otro lado, alimentamos el módulo aplicando la tensión de alimentación entre Vcc y GND del módulo. Existen modelos de 28BYJ-48 de tensiones nominales 5 o 12V. Deberéis aplicar la tensión que corresponde con vuestro motor.

Finalmente, conectamos los pines IN1, IN2, IN3 e IN4 a cuatro salidas digitales de Arduino. Al activar cada una de las señales activaremos el par Darlington correspondiente, provocando el encendido de la bobina.

La conexión, vista desde Arduino, quedaría de la siguiente manera.




```
// DECLARACIÓN DE VARIABLES
//definicion de pins
const int motorPin1 = 8;     // 28BYJ48 In1
const int motorPin2 = 9;     // 28BYJ48 In2
const int motorPin3 = 10;    // 28BYJ48 In3
const int motorPin4 = 11;    // 28BYJ48 In4

//definicion variables
int motorSpeed = 1200;   //variable para fijar la velocidad
int stepCounter = 0;     // contador para los pasos
int stepsPerRev = 4076;  // pasos para una vuelta completa
```

```
//tablas con la secuencia de encendido (descomentar la que necesiteis)
//secuencia 1-fase
//const int numSteps = 4;
//const int stepsLookup[4] = { B1000, B0100, B0010, B0001 };

//secuencia 2-fases
//const int numSteps = 4;
//const int stepsLookup[4] = { B1001, B1100, B0110, B0011 };

//secuencia media fase
const int numSteps = 8;
const int stepsLookup[8] = { B1000, B1100, B0100, B0110, B0010, B0011, B0001, B1001 };

// DECLARACIÓN DE FUNCIONES
void setOutput(int step){
  digitalWrite(motorPin1, bitRead(stepsLookup[step], 0));
  digitalWrite(motorPin2, bitRead(stepsLookup[step], 1));
  digitalWrite(motorPin3, bitRead(stepsLookup[step], 2));
  digitalWrite(motorPin4, bitRead(stepsLookup[step], 3));
}

void clockwise(){
  stepCounter++;
  if (stepCounter >= numSteps) stepCounter = 0;
  setOutput(stepCounter);
}

void anticlockwise(){
  stepCounter--;
  if (stepCounter < 0) stepCounter = numSteps - 1;
  setOutput(stepCounter);
}

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup(){
  //declarar pines como salida
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
  for (int i = 0; i < stepsPerRev * 2; i++)
  {
    clockwise();
    delayMicroseconds(motorSpeed);
  }
  for (int i = 0; i < stepsPerRev * 2; i++)
  {
    anticlockwise();
    delayMicroseconds(motorSpeed);
  }
  delay(1000);
}
```

MÓDULO RFID DE CONTROL DE ACCESOS

Los sistemas RFID (siglas de Radio Frequency IDentification, en español identificación por radiofrecuencia) estan compuestos por tarjetas, etiquetas, pegatinas y un sinfín de versiones de tags RFID con la capacidad de almacenamiento y recuperación de datos remoto.

El propósito fundamental de esta tecnología es la capacidad de identificar inequivocamente un dispositivo mediante un número de serie único que trae cada chip RFID. Pero además, los dispositivos RFID MIFARE disponen de una memoria interna que puede variar entre 1KByte y 4kByte.

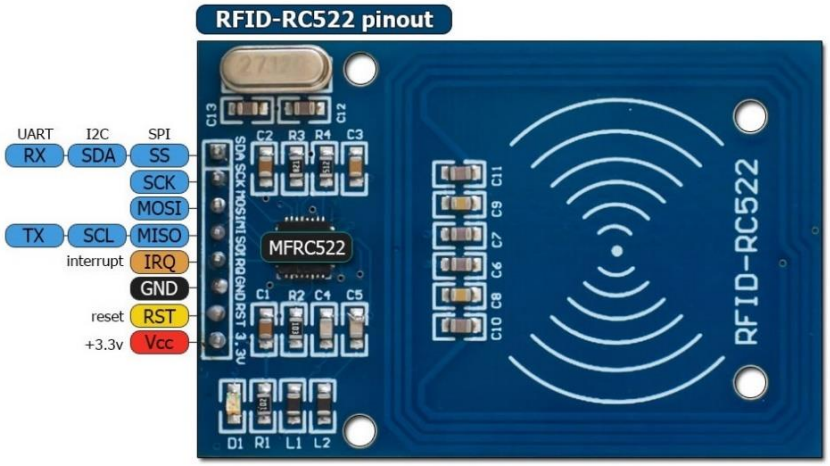
En esta práctica vamos a utilizar el Módulo Lector RFID-RC522 RF. Este módulo se controla a través del protocolo SPI, así como el protocolo UART. El RC522 utiliza un sistema avanzado de modulación y demodulación para todo tipo de dispositivos pasivos de 13.56Mhz.

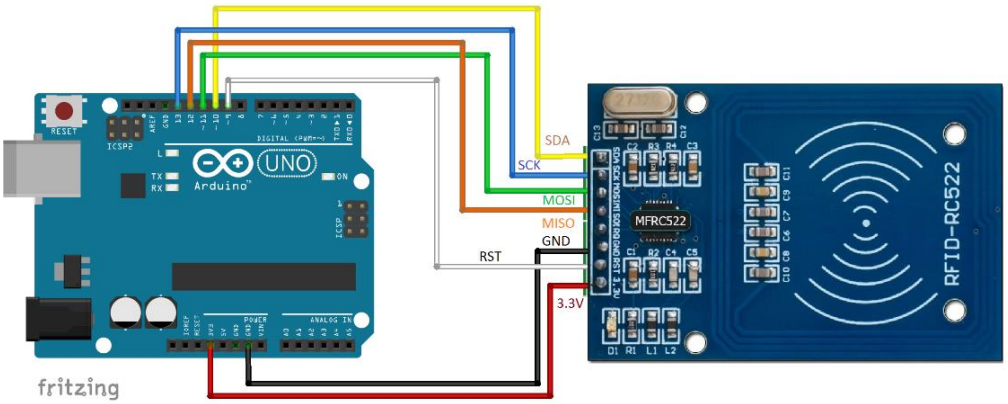
Las tarjetas disponen de un número de serie compuesto de 5 valores hexadecimales. Además, dispone de 64 bloques de memoria (0-63) donde podemos hacer lectura y/o escritura de datos. Cada bloque de memoria tiene la capacidad de almacenar hasta 16 Bytes (la grabación no la veremos en este curso)

CONECTAR EL MÓDULO RFID-RC522 RF

La conexión del módulo RFID-RC522 RF es muy sencilla. Utilizaremos los pines de arduino de comunicación SPI (diferente a la I2C que usamos en el módulo LCD)

MODULO RC522	ARDUINO
SDA	D10
SCK	D13
MOSI	D11
MISO	D12
IRQ	---
GND	GND
RST	D9
3.3V	3.3V





El uso de este módulo si supone un salto importante de dificultad. En internet existen multitud de código complejos debido al gran número de posibles tarjetas

diferente que existen en el mercado. Nosotros haremos un ejemplo sencillo de verificación:

- Leeremos una tarjeta
- Comprobaremos si está en una tabla de tarjetas permitidas
- Permitiremos o impediremos el acceso

Para ello instalaremos una librería sencilla llamada "RFID". Hay otras librerías más completas que permites también leer y escribir en las tarjetas pero no las utilizaremos en este curso.

```
// INCLUIMOS LAS LIBRERÍAS
#include <SPI.h>
#include <RFID.h>

// DECLARACIÓN DE VARIABLES Y OBJETOS
#define SS_PIN 10
#define RST_PIN 9
RFID rfid(SS_PIN,RST_PIN);

int led = 7;            // PIN indicación de acceso permitido
int serNum[5];
int clave_ok = 0;
bool access = false;
int cards[][5] = {
  {180,17,121,26,200},
  {200,27,221,46,592},
  {180,17,121,26,198},
  {37,30,255,176,116}
};                      // Lista de Array de Tarjetas permitidas
// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup(){

    Serial.begin(9600);     // Iniciamos puerto serie
    SPI.begin();            // Iniciamos protocolo SPI
    rfid.init();            // Iniciamos Objeto RFID

    pinMode(led, OUTPUT);   // iniciamos El PIN del led como salida
    digitalWrite(led, LOW);
    pinMode(power, OUTPUT);   // iniciamos El PIN del rele como salida
    digitalWrite(power, LOW);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
    if(rfid.isCard()){   // Detectamos la presencía de una tarjeta en el lector
        if(rfid.readCardSerial()){
            Serial.print(rfid.serNum[0]);
            Serial.print(" ");
            Serial.print(rfid.serNum[1]);
            Serial.print(" ");
            Serial.print(rfid.serNum[2]);
            Serial.print(" ");
```

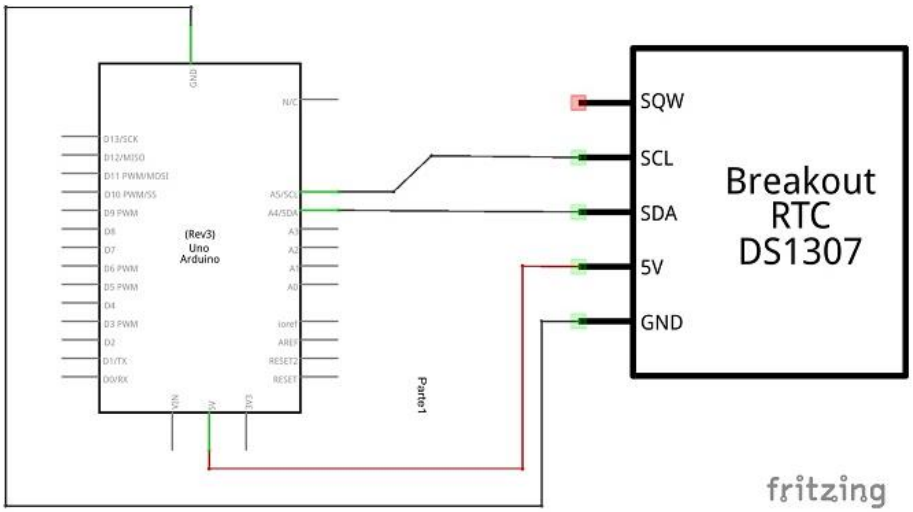
```
            Serial.print(rfid.serNum[3]);
            Serial.print(" ");
            Serial.print(rfid.serNum[4]);
            Serial.println("");
                  // Comprobamos cada uno de los valores del array de cards permitidas
            for(int x = 0; x <=(sizeof(cards)/2); x++){
              for(int i = 0; i < sizeof(rfid.serNum); i++ ){
                  if(rfid.serNum[i] == cards[x][i]) {
                      clave_ok++;
                      if(clave_ok==5) {
                        access = true;
                        break;
                      }
                      else access = false;
                   }
              }
          }

      }
    } // Cerramos la lectura de la tarjeta RFID

clave_ok=0;
       if(access){
          Serial.println("BIENVENIDO!");
           digitalWrite(led, HIGH);
           delay(2000);
           digitalWrite(led, LOW);
           access = false;

      } else {
           Serial.println("NO PERMITIDO!");
           digitalWrite(led, HIGH);
           delay(500);
           digitalWrite(led, LOW);
           delay(500);
           digitalWrite(led, HIGH);
           delay(500);
           digitalWrite(led, LOW);
       }

    rfid.halt(); //Ponemos el Módulo RFID en hibernación

}
```

MÓDULO DE CONTROL HORARIO RTC

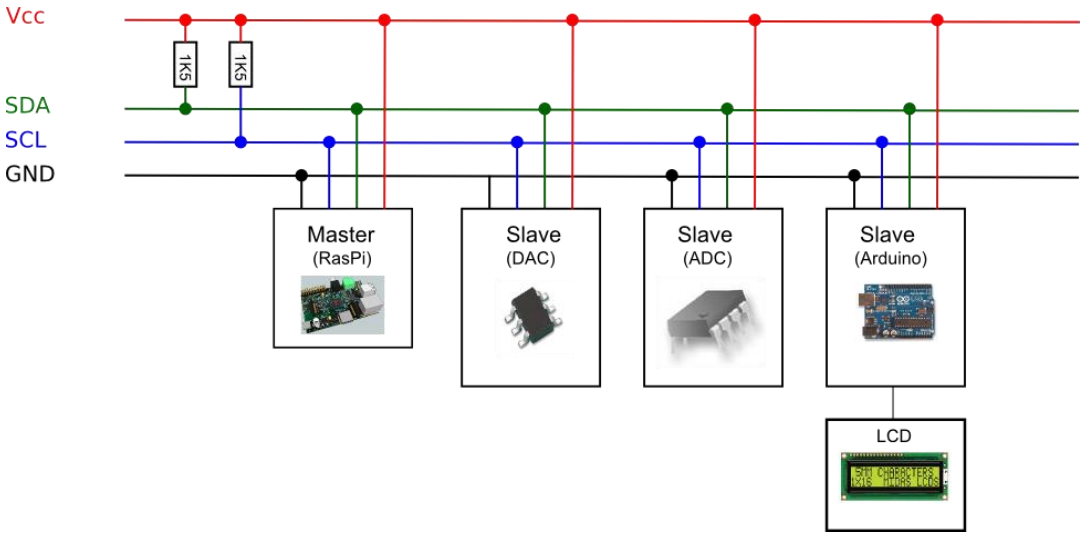
El reloj interneto de arduino tiene dos importantes limitaciones, se reinicia cada 50días aproximadamente y si el arduino se reinicia o se le corta la corriente el reloj se vuelve a cero empezando en el 1 de enero de 1970. Para solucionar esa situación utilizamos relojes digitales externos como los relojes Dallas DS1307 y DS3231.

CONECTAR EL MÓDULO RTC DS1307 o DS3231

La conexión con arduino de este módulo es muy sencilla. Usa el protocolo de comunicaciones I2C al igual que el display LCD que hemos ido usando hasta ahora. Por lo tanto, la conexión de pines es igual que para el display LCD



Es interesante saber que arduino puede controlar hasta 127 dispositivos diferentes a través del protocolo de comunicaciones I2C. Siempre y cuando las direcciones de los dispositivos sean diferentes entre sí.



Para funcionar con este módulo existen un elevado número de librerías diferentes en internet. Por su sencillez, nosotros vamos a optar por utilizar las siguiente: **"DS1307_DS3231_v1.4.7"** Esta librería ha sido desarrollado y publicada gratuitamente en su blog por Alberto Gil Tesa. Podeis acceder a ella y sus actualizaciónes:

https://giltesa.com/2012/09/02/libreria-gds1307-para-rtc

En esta práctica veremos uno de los ejemplo que vienen con la librería, con el único objetivo de saber como se configura y como se leen los datos del módulo RTC.

```
/*
* Ejemplo número 2
*/

#include <Wire.h>
#include <RTC.h>

RTC rtc(DST_ON);

void setup()
{
    Serial.begin(9600);
    while(!Serial);
    delay(1000);
}

void loop()
{

    // To set the clock from the IDE, enter the
    // date and the time following this format: 2014 10 14 22 49 18
    // Year, Month, Day, Hour24H, Minutes and Seconds

    if( Serial.available() > 0 )
    {
        int year    = Serial.parseInt();
        int month   = Serial.parseInt();
        int day     = Serial.parseInt();
        int hour24h = Serial.parseInt();
        int minutes = Serial.parseInt();
        int seconds = Serial.parseInt();
        rtc.setDateTime( year, month, day, hour24h, minutes, seconds );

        while( Serial.available() > 0 && Serial.read() != -1 );
    }

    Data d = rtc.getData();
    Serial.println( d.toString() );
    delay(1000);
}
```

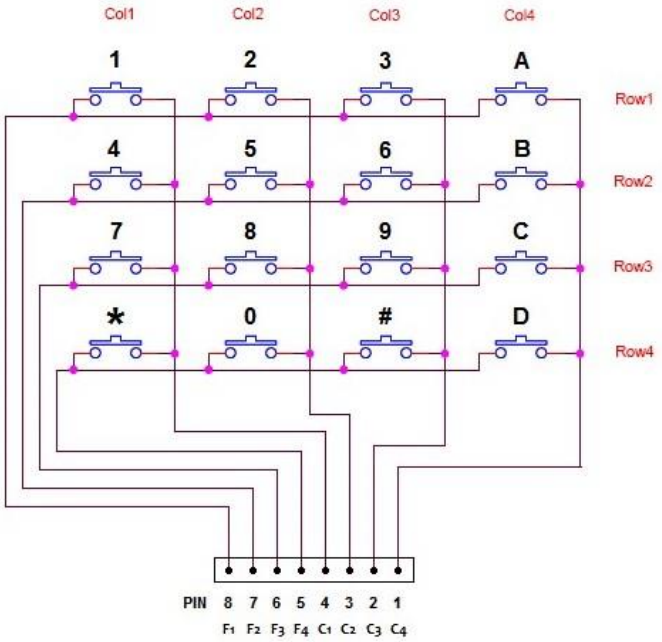
Una vez que hayamos puesto en hora el reloj podemos realizar una serie de funciones para las que está preparado esta librería.

1- Convertir los datos recibidos en un String para ser almacenandos (ex 5)
2- Comparar la datos recibidos. Mayor que, menos que, entre... (ex 8)

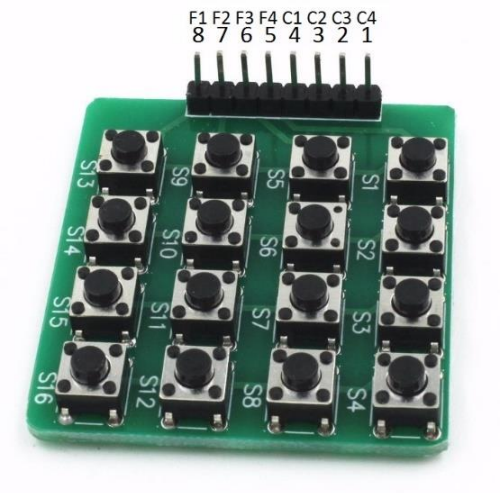
MÓDULO TECLADO MATRICIAL

La ventaja de utilizar este tipo de telcados es que se puede leer el estado de múltiples botones utilizando un número reducido de pines de nuestro Arduino. Así, por ejemplo, un teclado matricial de 4×4 que se compone de 16 pulsadores, solamente necesita 4 pines para las filas y otros 4 pines para las colúmnas. De esta forma se pueden leer las 16 teclas utilizando solamente 8 pines, en lugar de los 16 que serían necesarios utilizando otra configuración.

El sistema en el que se basa el funcionamiento de un teclado matricial es el multiplexado. Configuramos cuatro pines de nuestro microprocesador como entradas y otros cuatro como salidas. Conectamos las entradas a las columnas y las salidas a las filas (o viceversa porque el funcionamiento sería el mismo). Empezamos con las cuatro salidas en nivel bajo. Activamos la salida de la primera fila y leemos las cuatro entradas. Cada una de ellas nos informará del estado de los cuatro pulsadores de esa fila. La entrada 1 el pulsador de la columna 1, la entrada 2 el pulsador de la columna 2 y así sucesivamente. A continuación, desactivamos la salida 1 y activamos la 2. En este momento en las cuatro entradas podemos leer el estado de los pulsadores situados en la segunda fila. Haremos lo mismo con las salidas 3 y 4 para obtener el estado de los pulsadores situados en las filas 3 y 4.



Afortunadamente, no tenemos que preocuparnos de la programación de nuestro multiplexor, porque Arduino dispone de librerías para ello. Así que el uso de un teclado matricial es muy sencillo. Importamos la librería correspondiendo y solo debemos configurar los pines de las filas y columnas de nuestro arduino.



ARDUINO	D6	D7	D8	D9	D10	D11	D12	D13
TECLADO	1	2	3	4	5	6	7	8

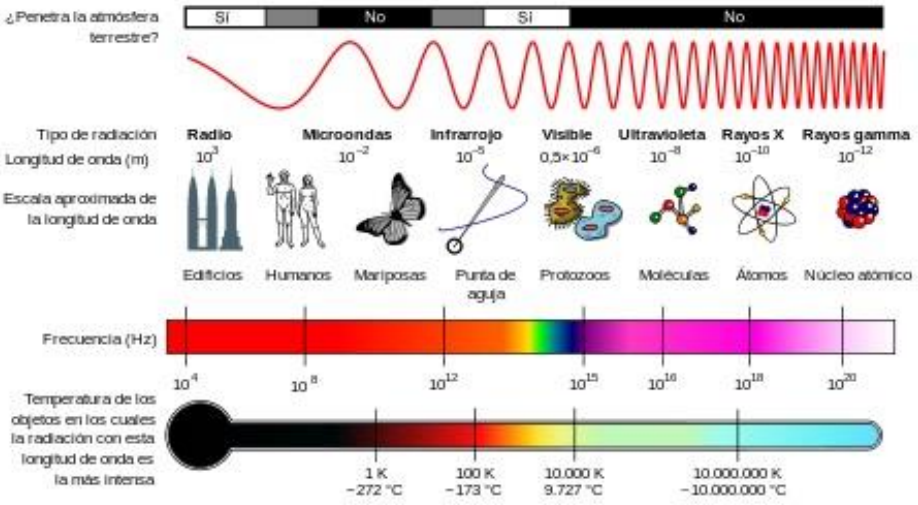
```
// INCLUIMOS LAS LIBRERÍAS QUE VAMOS A UTILIZAR
#include <Keypad.h>

// DECLARACIÓN DE VARIABLES
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
//define the cymbols on the buttons of the keypads
char hexaKeys[ROWS][COLS] = {
  {'0','1','2','3'},
  {'4','5','6','7'},
  {'8','9','A','B'},
  {'C','D','E','F'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {13, 12, 11, 10}; //connect to the column pinouts of the keypad

//initialize an instance of class NewKeypad
Keypad myKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup(){
  Serial.begin(9600);
}
// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop(){
  char Key = myKeypad.getKey();

  if (Key){
    Serial.println(Key);
  }
}
```

Una vez recogido el dato, podemos almacenarle en un array para ser comparado o compararte directamente con el dato obtenido, por ejemplo:

```
Int posición = 0;
Char CodigoSecreto = “25668”
If (key == CodigoSecreto[posicion])
{
Posicion ++;
}
If (posicion == 5)
{
// El código marcado el correcto
}
```

RECEPTOR IR CON MANDO A DISTANCIA

Los led infrarrojos se caracterizan por emitir radiación electromagnética, invisible al ojo humano, ya que su longitud de onda es mayor a la del espectro visible. Debemos tener en cuenta, que para que el receptor se comporte correctamente, el Tx led IR que usemos tiene que tener la longitud de onda a la cual es más sensible el sensor. La más común es 940nm.



COMO FUNCIONA LA COMUNICACIÓN IR

Existen multitud de dispositivos que utilizan tecnología infrarroja para control remoto, por ejemplo los mandos de las televisiones. Su funcionamiento se basa en la emisión de secuencias de pulsos de luz IR que transportan un código de N bits que identifica la tecla pulsada.

Existen muchos protocolos de comunicación infrarroja, nos vamos a centrar en el protocolo NEC, que es el que usan los mandos universales.

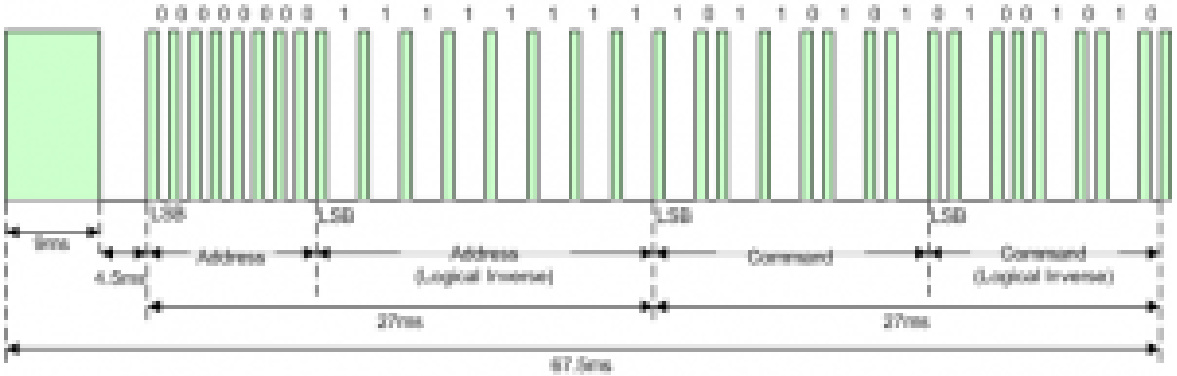
El protocolo NEC utiliza pulsos distancia para la transmisión de un mensaje. Este protocolo funciona a una frecuencia de 38kHz, es decir, los pulsos duran 562.5 µs. El mensaje se transmite de la siguiente forma:

- **0 lógico:** Envía un pulso de 562,5 µs, seguido de un espacio de 562,5 µs de duración (tiempo de transmisión de 1,125 ms)
- **1 lógico:** Envía un pulso de 562,5 µs seguido de un espacio de 1.6875 ms de duración (tiempo de transmisión de 2,25 ms)

Cuando pulsamos un botón de un mando, se envía la siguiente cadena de pulsos:

1- Una ráfaga de pulsos de 9 ms.
2- Un espacio 4,5 ms
3- La dirección para el dispositivo receptor (8 bits)
4- El inverso lógico de la dirección (8 bits) (para comprobar errores)
5- El comando (8 bits)
6- El inverso lógico del comando (8 bits)
7- Un pulso de 562.5 µs para indicar el final de la transmisión del mensaje.

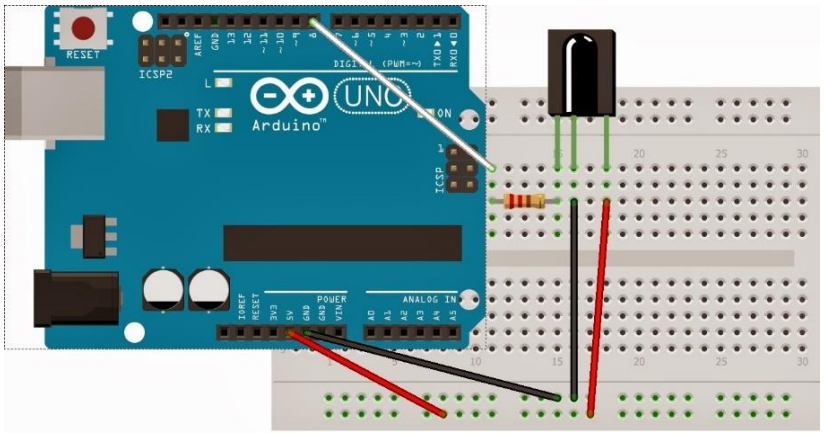
A continuación se ilustra con una imagen para una dirección 00000000 y un comando 10110101:



COMO CONECTAR EL MÓDULO CON ARDUINO

La conexión de este módulo con el arduino es muy sencilla. El módulo solo precisa de tres pines. Un pin digital para la comunicación y los dos pines de alimentación habituales Vcc (+5V) y GND.

Si deseamos conectar un led sensor IR sin el módulo pertinente debemos conectar una resistencia en serie de 220Ohm a la salida del Pin Vout.



Con la librería correspondiente el uso de las comunicaciones IR pueden ser un tema realmente sencillo. Existen una gran variedad de librerias en internet. Nosotro utilizaremos "IRremote".

```
//INCLUIMOS LIBRERÍAS
#include <IRremote.h>

//INICIALIZAMOS LAS VARIABLES Y LOS OBJETOS
int IRpin = 11;
IRrecv irrecv(IRpin);
decode_results results;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Empezamos a recibir los datos
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
  if (irrecv.decode(&results))
    {
      Serial.println(results.value, DEC); // Print por Serial 'results.value'
      irrecv.resume();   // Dejamos el sistema a la espera del próximo dato
    }
}
```

Práctica: Controlar el encendido y apagado de las entradas digitales

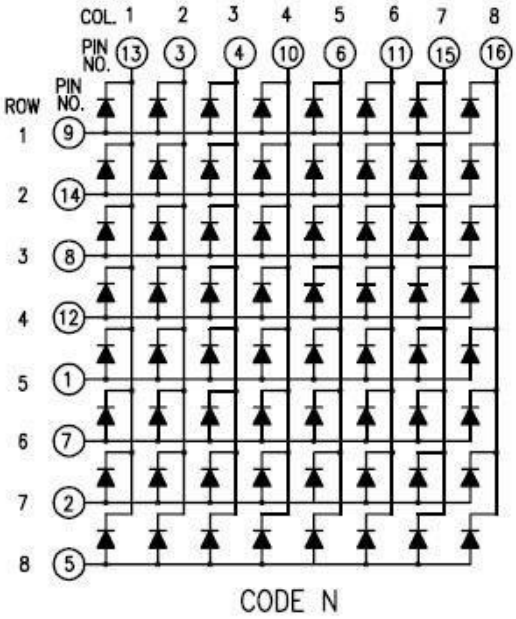
MÓDULO MATRIZ DE LEDS 8X8

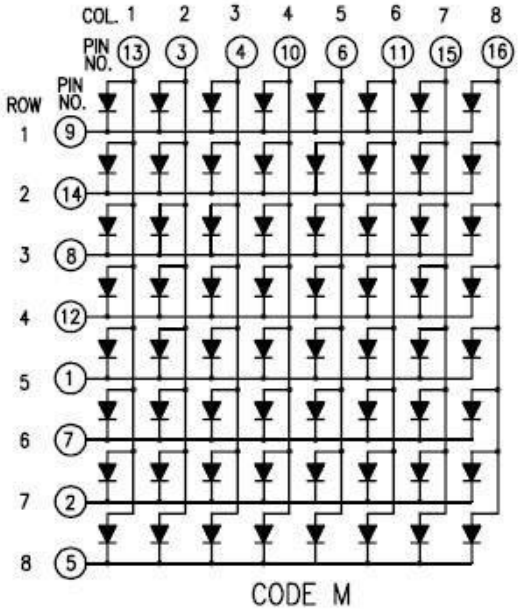
Una matriz LED es un display formado por **múltiples LED en distribución rectangular**. Existen distintos tamaños, siendo el más habitual los cuadrados de 8×8 LED.

Podemos combinar varios módulos para formar un display mucho mayor. En estos display podemos mostrar textos, dibujos o animaciones, como desplazar un texto (scroll).

COMO FUNCIONA UNA MATRIZ DE LED

Una matriz de LED está formado por diferentes LED, **cableados de forma conjunta por filas y columnas**. Podemos encender un LED determinado de la matriz aplicando correctamente los valores HIGH y LOW a su respectiva fila y columna.





Si aplicamos valores HIGH y LOW a varias filas y columnas de forma simultánea se encenderán todos los LED de las intersecciones. De esta forma, resulta casi imposible generar gráficos complejos.

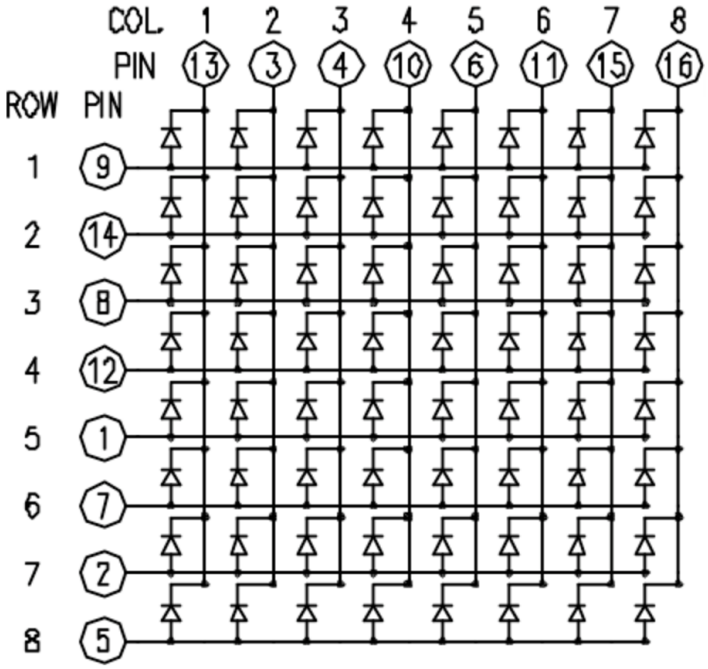
Para poder hacer gráficos el procedimiento consiste en hacer un barrido por filas (o columnas). Encendemos todos los LED de una única fila (o columna), y a continuación cambiamos a la siguiente. Sólo una fila (o columna) está encendida cada instante pero, al

hacerlo rápidamente, el efecto en nuestra visión es ver toda la imagen formada de forma simultánea.

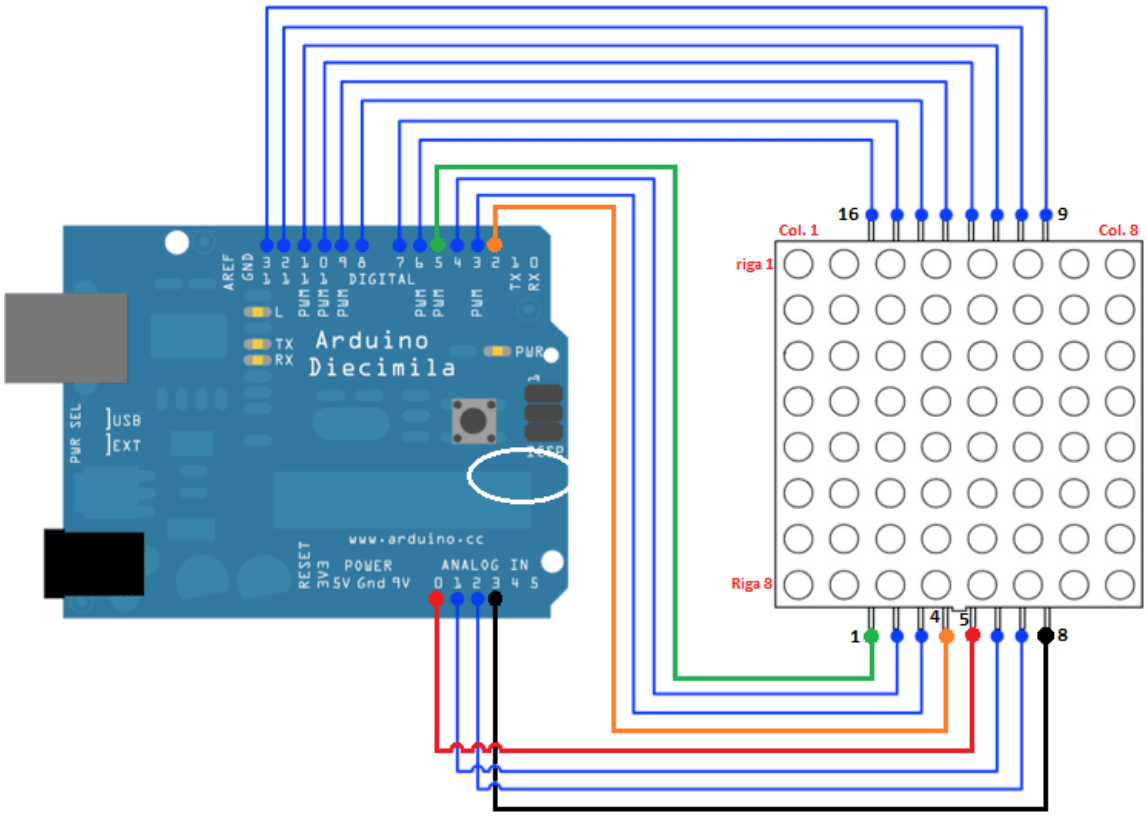
Encender una matriz de 8×8 LED requeriría 16 señales digitales y un trabajo constante del procesador para refrescar la imagen. Eso es una cantidad enorme de recursos para cualquier autómata, que estaríamos mal aprovechando para simplemente encender un display. Por este motivo, en la práctica **empleamos un controlador como el MAX7219** que está especialmente diseñado para encender displays de 7 segmentos y matrices de led, y libera a nuestro procesador para hacer tareas mucho más valiosas.

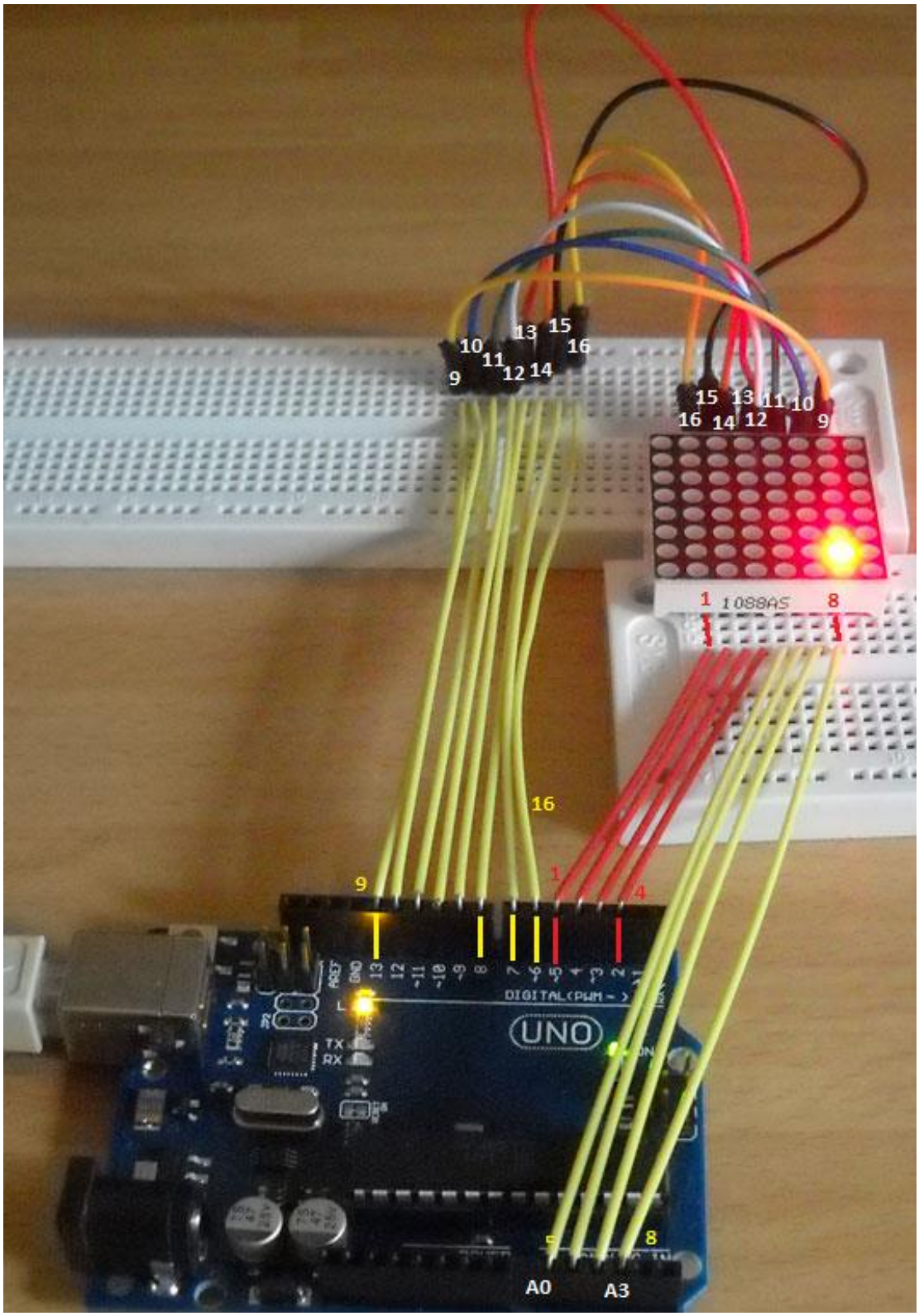
CONECTAR LA MATRIZ LED 8X8

En esta práctica, trantandose de un ejercicio educativo utilizaremos la matriz de led desde un punto de vista teórico, sin tener encuenta la cantidad de pines utilizados.



Pin Dot Matrix	Pin Arduino (D)igitale / (A)nalogico	(R)iga / (C)olonna Dot Matrix
1	D5	R5
2	D4	R7
3	D3	C2
4	D2	C3
5	A0	R8
6	A1	C5
7	A2	R6
8	A3	R3
9	D13	R1
10	D12	C4
11	D11	C6
12	D10	R4
13	D9	C1
14	D8	R2
15	D7	C7
16	D6	C8


Arduino
Diecimila
DIGITAL
POWER
ANALOG IN
www.arduino.cc
USB
EXT
PWR SEL
Col. 1
Col. 8
riga 1
Riga 8
16
9
1
8
4
5


10
11
9
12
13
14
15
16
15
16
14
13
12
11
10
9
1088AS
1
8
16
9
1
4
DIGITAL(PWM~)
UNO
TX
RX
L
5
8
A0
A3

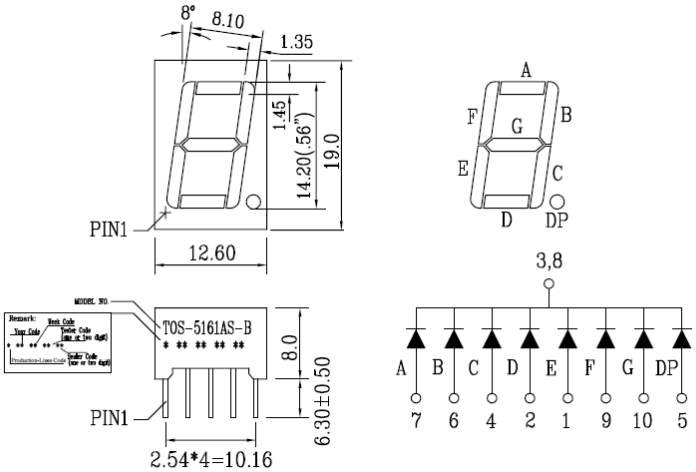
```
// DECLARACIÓN DE VARIABLES
// En el siguiente array pin[0..16] (la posición 0 es ficticia y sirve para iniciar el
indice de la matriz en 1)
// Se trata de esquematizar la conexión de pines en la matriz LED
// Tenga presente que los pines 14 ... 17 corresponden respectivamente a:
//    A0 => 14-esimo pin de arduino
//    A1 => 15-esimo pin de arduino
//    A2 => 16-esimo pin de arduino
//    A3 => 17-esimo pin de arduino
int pins[17]= {-1, 5, 4, 3, 2, 14, 15, 16, 17, 13, 12, 11, 10, 9, 8, 7, 6};

int cols[8] = {pins[13], pins[3], pins[4], pins[10], pins[06], pins[11], pins[15],
pins[16]};
int rows[8] = {pins[9], pins[14], pins[8], pins[12], pins[1], pins[7], pins[2],
pins[5]};

// DECLARACIÓN DE FUNCIONES
void AllLedOFF()  // Creamos una función que apaga todos los LEDS
{
  for (int i = 1; i <= 8; i++)
  {
        digitalWrite(cols[i - 1], LOW);
        digitalWrite(rows[i - 1], HIGH);
  }
}

void LedON(int r, int c)      //Encendemos el LED en la posición fila R y columna R
{
  AllLedOFF();
  digitalWrite(cols[c - 1], HIGH);
  digitalWrite(rows[r - 1], LOW);
  delay(100);
}

void AllLedON()     // Creamos una función que enciende todos los LEDS
{
  for (int i = 1; i <= 8; i++)
  {
        digitalWrite(cols[i - 1], HIGH);
        digitalWrite(rows[i - 1], LOW);
  }
  delay(1000);
  AllLedOFF();
}

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  // Definimos todos los pin de Arduino como pines de salida
  for (int i = 1; i <= 16; i++)
      pinMode(pins[i], OUTPUT);
  // Ponemos todos los LED la matriz apagados
  AllLedOFF();
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
```

```
void loop()
{
  int i, c, r;

  // Vamos encendiendo los LED por columnas de izquierda a derecha
  for (c=1; c<=8; c++)
  {
    for (r=1; r<=8; r++)
       digitalWrite(rows[r - 1], LOW);
    digitalWrite(cols[c - 1], HIGH);
    delay(300);
    digitalWrite(cols[c - 1], LOW);
    for (r=1; r<=8; r++)
       digitalWrite(rows[r - 1], HIGH);
  }

  // Vamos enciando los LED por filas de abajo a arriba
  for (r=1; r<=8; r++)
  {
    for (c=1; c<=8; c++)
       digitalWrite(cols[c - 1], HIGH);
    digitalWrite(rows[r - 1], LOW);
    delay(300);
    digitalWrite(rows[r - 1], HIGH);
    for (i=1; i<=8; i++)
      digitalWrite(cols[i - 1], LOW);
  }

  // Encendemenos cada led independiente desde la primera posición hasta la 64
  for (r=1; r<=8; r++)
    for (c=1; c<=8; c++)
      LedON(r,c);
  AllLedOFF();
  // AllLedON();
}
```

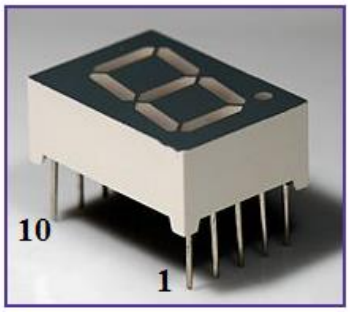
DISPLAY 1 DIGITO 7 SEGMENTOS

Estos display leds estan formados por 7 lineas que pueden muestrar digitos del 0 a 9. A cada uno de esos segmentos se les representra con una letra a, b, c, e, f, g y aparte un punto.

el display que vamos a utilizar e un 5161AS que tiene la siguiente distribución de patillaje:

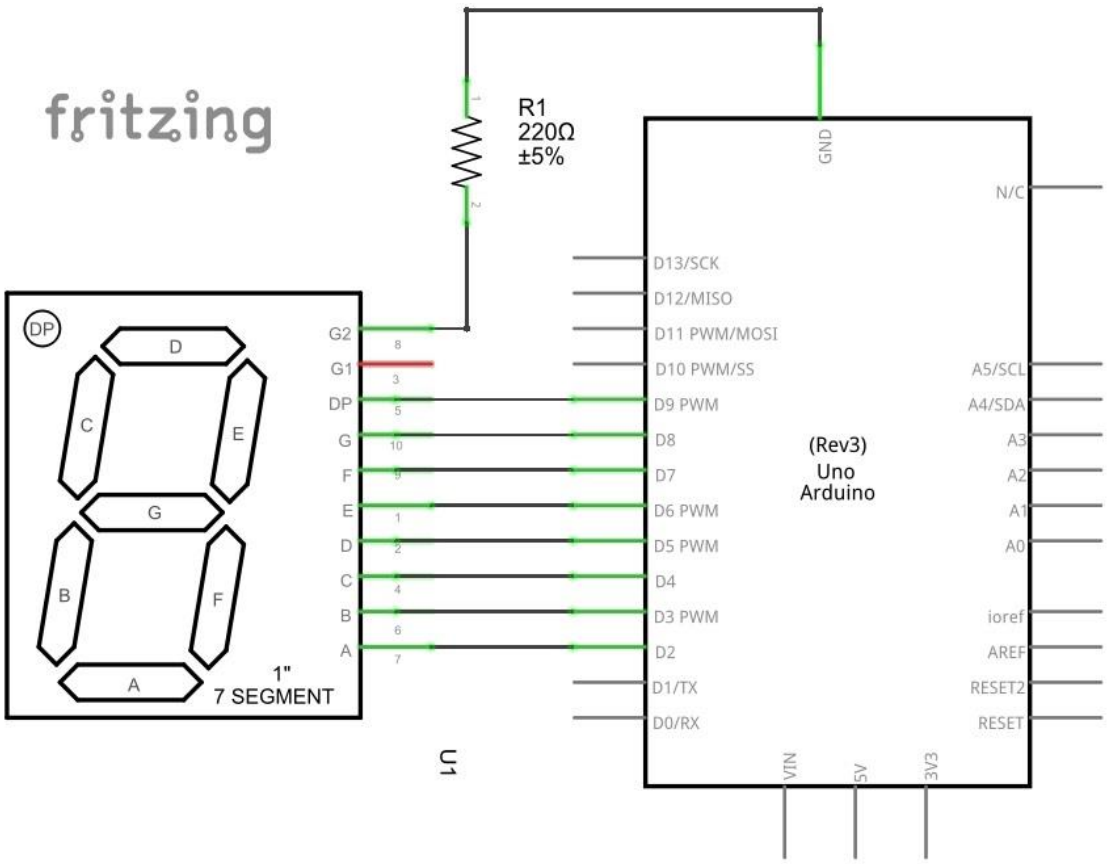


Para saber cual es el pin 1, poner el display de modo que viendo los numeros el punto decimal quede abajo a vuestra derecha. El pin numero 1 es en la parte inferior el primero por la izquierda, y luego ir contando en el sentido contrario a las agujas del reloj.

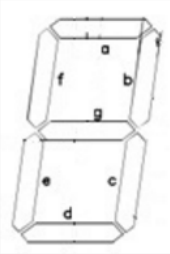


CONECTAR EL DISPLAY A ARDUINO

La conexión es como si hiciesemos un circuito con 8 diodos. Con una resistencia colocada en serie.



Para mostrar un dígito en el display debemos encender y apagar diferentes leds en el display. Esta lista tabla muestra la combinación de leds que deben estar encendidos y apagados para mostrar cada dígito:

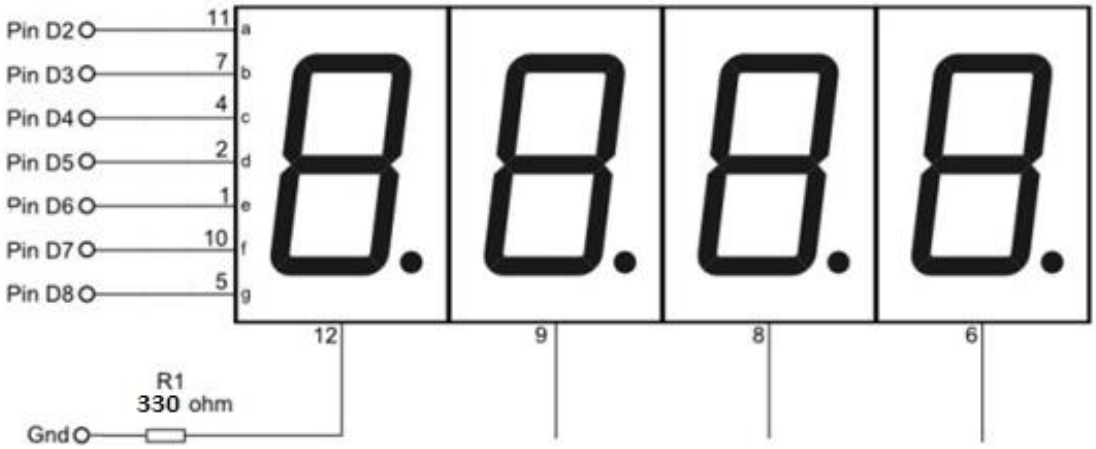
	DIGITO	A	B	C	D	E	F	G	PUNTO
	0	1	1	1	1	1	1	0	0
	1	0	1	1	0	0	0	0	0
	2	1	1	0	1	1	0	1	0
	3	1	1	1	1	0	0	1	0
	4	0	1	1	0	0	1	1	0
	5	1	0	1	1	0	1	1	0
	6	1	0	1	1	1	1	1	0
	7	1	1	1	0	0	0	0	0
	8	1	1	1	1	1	1	1	0
	9	1	1	1	0	0	1	1	0

```
// DECLARACIÓN DE VARIABLES
byte  Digit[10][8] =                // Arduino UNO va muy justo de memoria. Por eso lo
{                                   // definimos como byte y no como int
   { 1,1,1,1,1,1,0,0 },    // 0
   { 0,1,1,0,0,0,0,0 },    // 1
   { 1,1,0,1,1,0,1,0 },    // 2
   { 1,1,1,1,0,0,1,0 },    // 3
   { 0,0,1,0,0,1,1,0 },    // 4
   { 1,0,1,1,0,1,1,0 },    // 5
   { 1,0,1,1,1,1,1,0 },    // 6
   { 1,1,1,0,0,0,0,0 },    // 7
   { 1,1,1,1,1,1,1,0 },    // 8
   { 1,1,1,0,0,1,1,0 }     // 9
};

// DECLARACIÓN DE FUNCIONES
void Display(int N)
   {
       for (int i= 0 ; i<8 ; i++)
            {   int valor = Digit[N][i] ;
                int pin = i+2;
                digitalWrite(pin , valor) ;
            }
   }

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
   {   for (int i= 2; i<11; i++)
            pinMode(i, OUTPUT);
   }


// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
   {
       for ( int k=0 ; k<10 ; k++)    // Llama a Display para k de 0 a 9, los digitos
         {  Display(k);
            delay(1000);
         }
   }
```

DISPLAY 4 DIGITOS 7 SEGMENTOS

El display de 4 dígitos con 7 segmentos usa 8 pines para iluminar los 7 segmentos más el puntos y otros 4 pines que indican cual es el dígito que vamos a encender.

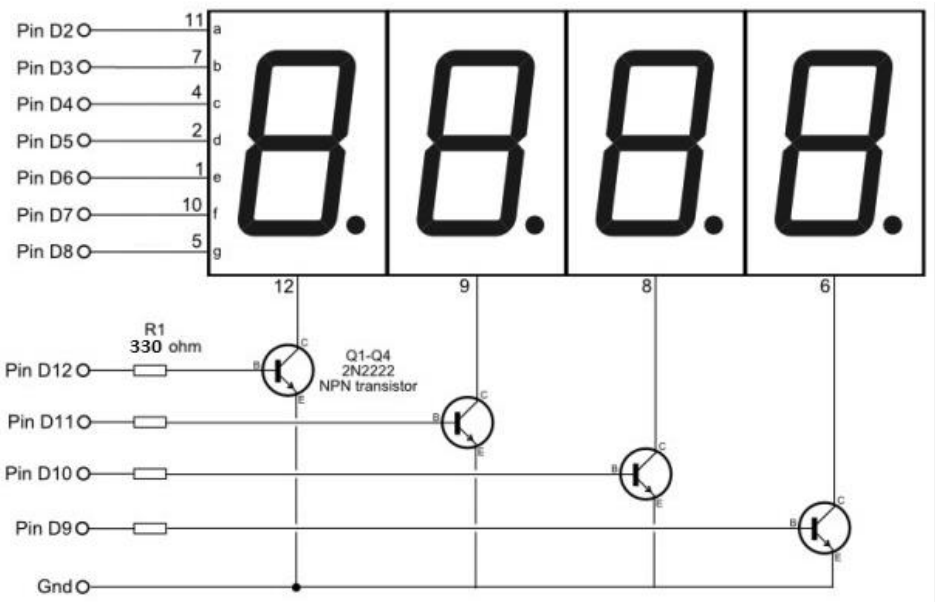
Este diagrama corresponde al display (5461AS)



El consumo típico de cada segmento, (como de cada LED) es de entre 15 y 20mA. Como son 8 segmentos cuando escribamos el 8 más el punto, el consumo será de 8 x 15 = 120 mA y pudiendo llegar a 160mA. Muy por encima de los 40 mA que un pin de Arduino puede suministrar..

La solución es utilizar transistores 2N2222 para permitir cerrar el circuito de cada dígito. Poniendo a LOW los pines 9, 10, 11, y 12 los transistores están al corte, y ningún digito puede iluminarse. Si ponemos tensión en el pin 12, su transistor se satura y permite la salida a Ground del primer digito que encenderá los segmentos que le indiquen los pines 2 al 8.

Ahora ponemos en LOW el pin 12 y en HIGH el 11. Los valores de los segmentos iluminaran ahora el segundo digito. Y luego el terceo y el cuarto. El único truco es hacerlo suficientemente rápido para que el cambio no sea perceptible al ojo humano.




```
// DECLARACIÓN DE VARIABLES
byte  Digit[10][8] =                // Arduino UNO va muy justo de memoria. Por eso lo
{                                   // definimos como byte y no como int
   { 1,1,1,1,1,1,0,0 },    // 0
   { 0,1,1,0,0,0,0,0 },    // 1
   { 1,1,0,1,1,0,1,0 },    // 2
   { 1,1,1,1,0,0,1,0 },    // 3
   { 0,0,1,0,0,1,1,0 },    // 4
   { 1,0,1,1,0,1,1,0 },    // 5
   { 1,0,1,1,1,1,1,0 },    // 6
   { 1,1,1,0,0,0,0,0 },    // 7
   { 1,1,1,1,1,1,1,0 },    // 8
   { 1,1,1,0,0,1,1,0 }     // 9
};

// DECLARACIÓN DE FUNCIONES
void Display(int pos, int N)
   {
       digitalWrite(9 ,LOW);        // Apaga todos los digitos
       digitalWrite(10,LOW);
       digitalWrite(11,LOW);
       digitalWrite(12,LOW);

      for (int i= 0 ; i<8 ; i++)    // Esto no cambia de la session anterior
            digitalWrite(i+2 , Digit[N][i]) ;

      digitalWrite(pos + 9, HIGH);      // Enciende el digito pos
  }
void CalculaDigitos( int Num)
   {
    //2386 % 10 = 6          Porque el %  nos devuelve el resto de dividir por 10, o
sea 6.
    //2386 % 100 = 86        Ahora hacemos la división entera por 10 o sea 8,los
decimales se tiran
    //2386 % 1000 = 386      386 / 100 = 3
    //2386  / 1000 = 2       Directamente, para eso se inventó la división entera.
```

```
      int Digit0 = Num %10 ;
      int Digit1 = (Num % 100) / 10 ;
      int Digit2 = (Num % 1000) / 100 ;
      int Digit3 = Num  / 1000)  ;

      Display(3 , Digit3);
      Display(2 , Digit2);
      Display(1 , Digit1);
      Display(0 , Digit0);
   }

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
    {
      for (int i= 2; i<13; i++){
        pinMode(i, OUTPUT);
      }
    }


// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
   {
       int n = millis() / 1000 ;       // Lo pasamos a segundos
       int segundos = n % 60  ;
       int minutos =  n / 60  ;

       int k = minutos * 100 + segundos ;
       CalculaDigitos(k) ;
   }
```

PROYECTO: SISTEMA DE CONTROL DE ACCESO RFID

DESCRIPCIÓN:

Con los módulos que hemos visto hasta ahora te dejo como reto diseñar un sistema de control de accesos. El desarrollo podría usar los siguientes componentes:

- Arduino
- Sensor PIR
- Buzzer activo o pasivo
- Relé
- Módulo RFID
- Display LCD 1602 con i2c
- Móduto RTC
- Sensor tilt
- Matriz de pulsadores
- Led ROJO y VERDE
- Cualquier otro...

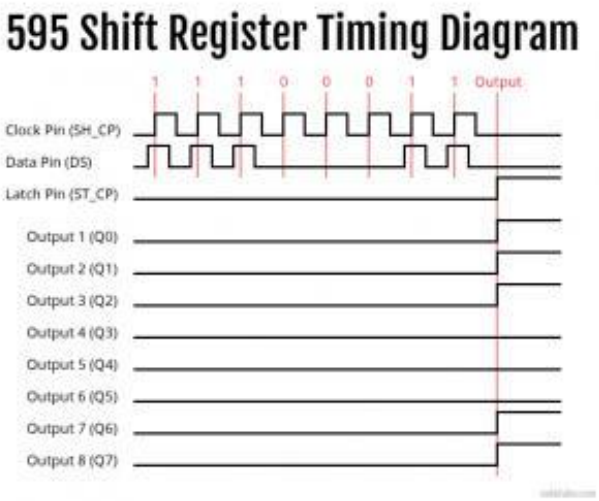
PRÁCTICAS AVANZADAS

REGISTRO DE DESPLAZAMIENTO 74HC595 8 BITS

El registro de desplazamiento 74HC595 es un sistema Serial-in – Parallel-out de 8bits que convierte los datos serie a paralelo, es decir, Con 3 pines (Clock, Data y Latch) se puden controlar 8 salidas.

COMO FUNCIONA

Los registros de desplazamiento funcionan mediante la comunicación sincrona. Es decir, disponemos de un reloj (Clock pin), un pin que utilizamos para enviar los datos en serie (Data pin) y cuando los 8 bits de datos son leidos, un tercer pin (Latch pin) escribe los datos en los pines de salida.



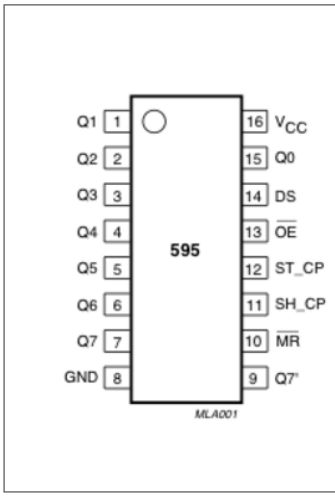
El arduino envía la señal de reloj y el registro de desplazamiento va leyendo los datos (1 1 1 0 0 0 1 1) en el flanco de subida. Cada uno de esos datos se distribuyen a los pines de salida de Q0 hasta el Q7, pero se activan hasta que el pin de Latch se activa (también por flanco de subida)

CONECTAR EL SHIFT-REGISTER

En esta práctica utilizaremos 8 leds y 8 resistencias de 220Ohm. Como se muestra en la imagen siguiente, se debe conectar el catodo de cada led (pin más corto) a GND y el anodo (pin más largo) de cada led a la salida que le corresponda en el registro de desplazamiento.

Usamos una función especial incluida en el IDE de arduino: shiftOut(dataPin, clockPin, bitOrder, value) con las siguientes características:

- dataPin: el pin donde están conectados los datos del 74hc595 (int)
- clockPin: el pin del reloj (int)
- bitOrder: el orden de los; se puede elegir entre MSBFIRST o LSBFIRST. (Primero el Bit más significativo, o, Primero el Bit menos significativo)
- value: el dato que quieremos envíar. (byte)



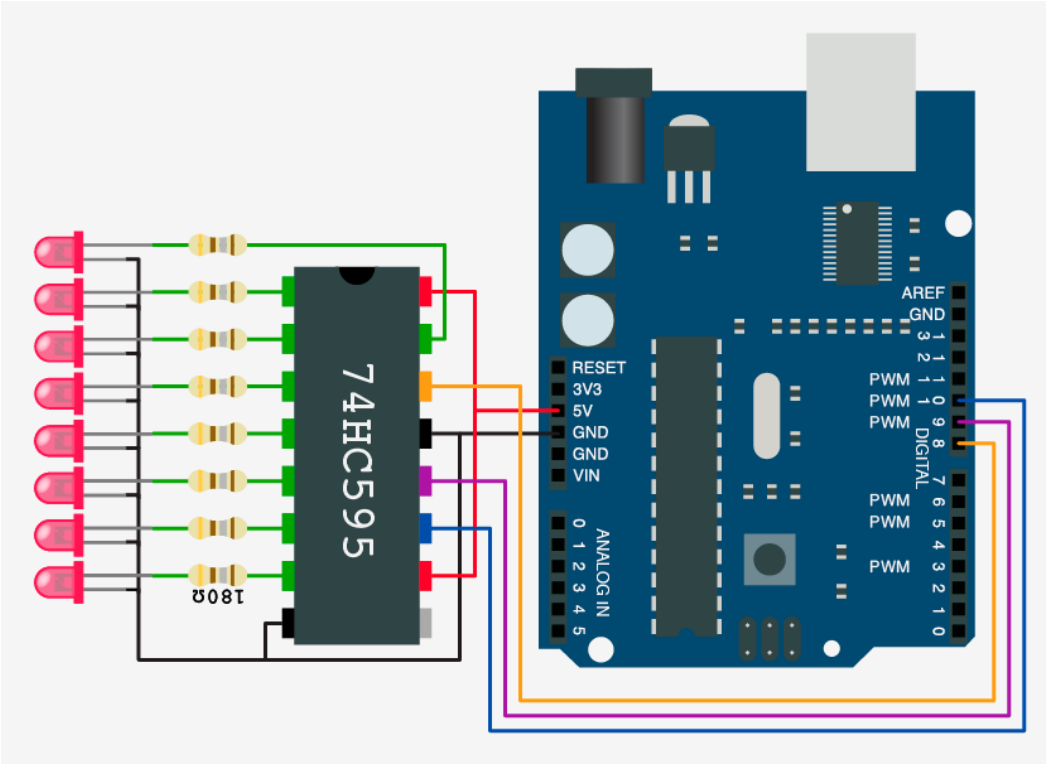
PINS 1-7, 15	Q0 " Q7	Output Pins
PIN 8	GND	Ground, Vss
PIN 9	Q7"	Serial Out
PIN 10	MR	Master Reclear, active low
PIN 11	SH_CP	Shift register clock pin
PIN 12	ST_CP	Storage register clock pin (latch pin)
PIN 13	OE	Output enable, active low
PIN 14	DS	Serial data input
PIN 16	Vcc	Positive supply voltage

Vamos a ver 2 ejemplos básicos.

1- Hola Mundo

```
// DECLARACIÓN DE VARIABLES
// Pin conectado a ST_CP del 74HC595
int latchPin = 12;
// Pin conectado a to SH_CP del 74HC595
int clockPin = 11;
// Pin conectado a to DS del 74HC595
int dataPin = 14;
// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  // Establecer los pines para que pueda controlar el registro de desplazamiento
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
  // Contar desde 0 a 255 y se muestra el número en los LEDs
  for (int numberToDisplay = 0; numberToDisplay < 256; numberToDisplay++) {
    // Poner el latchPin en LOW para apagar todos los leds
    digitalWrite(latchPin, LOW);
```

```
    // Los leds no cambian mientras se están enviado los datos:
    shiftOut(dataPin, clockPin, MSBFIRST, numberToDisplay);

    //Poner el LatchPin en valor HIGH y los pin se encienden:
    digitalWrite(latchPin, HIGH);
    // pausa antes del siguiente valor
    delay(500);
  }
}
```



2- A través del serial

```
// DECLARACIÓN DE VARIABLES
// Pin conectado a ST_CP del 74HC595
int latchPin = 12;
// Pin conectado a to SH_CP del 74HC595
int clockPin = 11;
// Pin conectado a to DS del 74HC595
int dataPin = 14;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup() {
  // Establecer los pines para que pueda controlar el registro de desplazamiento
  pinMode(latchPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("reset");
}
```

```
// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop() {
  if (Serial.available() > 0) {
    // En ASCII los valores de '0' hasta '9' caracteres son
    // representados por los valores 48 hasta el 57
    // por lo que para obtener el valor numérico de 0 a 9 se le debe restar 48
    int bitToSet = Serial.read() - 48;

  // Escribimos los datos en el registro de desplazamiento
    registerWrite(bitToSet, HIGH);
  }
}


void registerWrite(int whichPin, int whichState) {
  byte bitsToSend = 0;

// Poner el latchPin en LOW para apagar todos los leds
  digitalWrite(latchPin, LOW);

// Convertimos el número en bits
  bitWrite(bitsToSend, whichPin, whichState);

// Los leds no cambian mientras se están enviado los datos:
  shiftOut(dataPin, clockPin, MSBFIRST, bitsToSend);

//Poner el LatchPin en valor HIGH y los pin se encienden:
  digitalWrite(latchPin, HIGH);

}
```

MÓDULO GIROSCOPIO MPU-6050

EL MPU6050 es una unidad de medición inercial o IMU (Inertial Measurment Units) de 6 grados de libertad (DoF) pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc.

Con el acelerómetro podemos hacer mediciones indirectas como por ejemplo si integramos la aceleración en el tiempo tenemos la velocidad y si la integramos nuevamente tenemos el desplazamiento, necesitando en ambos casos la velocidad y la posición inicial respectivamente.

SENSOR GIROSCÓPICO

Un giroscopio es un dispositivo que funciona para miden velocidades angulares basándose en el mantenimiento del impulso de rotación. Si intentamos hacer girar un objeto que está girando sobre un eje que no es el eje sobre el que está rotando, el objeto ejercerá un momento contrario al movimiento con el fin de preservar el impulso de rotación total.

El giroscopio muestra el cambio de rango en rotación en sus ejes X, Y y Z.

ACELERÓMETRO

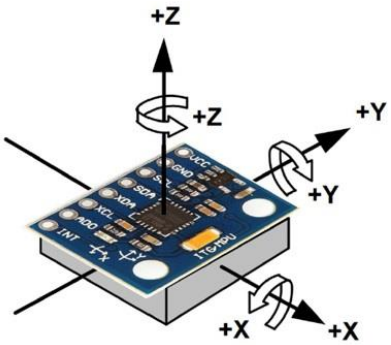
Mide la aceleración, inclinación o vibración y transforma la magnitud física de aceleración en otra magnitud eléctrica que será la que emplearemos en los equipos de adquisición estándar. Los rangos de medida van desde las décimas de grado, hasta los miles de grado.

COMO CONECTAR ARDUINO

La conexción con arduino es muy sencilla. Este módulo utiliza comunicación I2C y su dirección es 0x68. Conectando el pin AD0 se puede variar la dirección de 0x68 a 0x69.

MPU6050	Arduino Uno, Nano, Mini	Arduino Mega, DUE	Arduino Leonardo
VCC	5V	5V	5V
GND	GND	GND	GND
SCL	A5	21	3
SDA	A4	20	2

Este recuerso se utiliza para poder conectar varios acelerómetros al mismo host I2C.




```
#include<Wire.h>
const int MPU_addr=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}

void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  //equation for
temperature in degrees C from datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.println(GyZ);
  delay(333);
}
```

Si desea saber como medír la aceleración, el ángulo de inclinación, calibrar el sensor... y más aplicaciones... puedes consultar:

Inven.es

http://www.naylampmechatronics.com/blog/45_Tutorial-MPU6050-Aceler％C3％B3metro-y-Giroscopio.html

GUARDAR Y LEER DATOS EN LA MEMORIA EEPROM DE ARDUINO

Cada modelo de placa posee una EEPROM distinta. Aunque en casi cualquier tipo de proyecto te va a sobrar espacio con independencia de tu Arduino, en la siguiente imagen se muestran las diferentes capacidades de las memorias EEPROM.

Puedes programar la memoria EEPROM de tu Arduino fácilmente utilizando la librería EEPROM.h (podrás encontrarla por defecto en la IDE).

En la memoria de tu Arduino **debes trabajar por direcciones y byte a byte tanto para leer como para escribir**. Esto significa que para una memoria de 1kB tendrás desde la dirección 0 hasta la 999 y podrás utilizar valores de 0 a 255. En caso de que quieras guardar valores mayores tendrás que dividirlos por bytes.

Placa Arduino	Memoria EEPROM (KB)
UNO	1
DUE	-
Leonardo	1
Mega	4
Micro	1
Mini	1
Nano	1
Ethernet	1
Esplora	1
Bluetooth	1
Fio	1
Pro Mini	1
Lilypad	1

ESCRIBIR Y LEER UN SOLO BYTE

```
// DECLARACIÓN DE LIBRERIAS
#include <EEPROM.h>

// DECLARACIÓN DE VARIABLES
```

```
//Se crea una variable con el valor de la posición de memoria
//en la que se va a almacenar el byte.
int Direccion = 0;
//Se crean una variables para leer los valores de la memoria EEPROM
byte Val1;
byte Val2;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
Serial.begin(9600);
//Se almacena la información a guardar en un byte.
byte Informacion = B11001;// La "B" indica que el formato es binario
//También es posible almacenar la información leída a través de un
//sensor. Hay que tener en cuenta que el valor máximo de una variable
//tipo int es de 1023, mientras que el mayor valor que se puede almacenar
//en un solo byte es 255. Por tanto, se divide el valor resultante entre 4
//y a la hora de utilizar el valor se vuelve a multiplicar por 4. En el
//proceso se pierde precisión.
// need to divide by 4 because analog inputs range from
int Valor = analogRead(0) / 4;
//Se almacenan consecutivamente los valores anteriores.
EEPROM.write(Direccion, Informacion);
EEPROM.write(Direccion+1, Valor);
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL)
void loop()
{
Val1 = EEPROM.read(Direccion);
Val2 = EEPROM.read(Direccion+1);
Serial.print("En la dirección ");
Serial.print(Direccion);
Serial.print(" se encuentra la información: ");
Serial.print(Val1, DEC);// DEC para datos en decimal.
delay(100);
Serial.print("En la dirección ");
Serial.print(Direccion+1);
Serial.print(" se encuentra la información: ");
Serial.print(Val2, DEC);// DEC para datos en decimal.
delay(100);
}
```

ESCRIBIR VARIOS BYTES

```
//Se utiliza un elemento de tipo long (contiene 4 bytes).
long Dato = 123456789;

//Se almacena el dato en una variable auxiliar.
long AuxDato = Dato;

//Se define un byte para almacenar la información en la EEPROM.
byte B;
```

```
//La instrucción sizeof() devuelve el número de bytes de una variable
for (int i = 0; i < sizeof(Dato); i++){

//Se guarda el byte más significativo
B = highByte(AuxDato);
EEPROM.write(i, B);
//Se desplazan 8 bits hacia la izquierda de modo que los 8 primeros
//bits se pierden y los 8 siguientes pasan a ser los primeros
AuxDato = AuxDato << 8;
}
```

COMPONER VARIOS BYTES

```
//Variable que almacenará el resultado final.
long Dato = 0;

//Variable auxiliar (utilizada al mismo tiempo como byte).
long B;

for (int i = sizeof(Dato); i > 0 ; i--){

//Se almacena el valor de la EEPROM en los bits menos significativos.
B = EEPROM.read(i-1);

//Se desplazan los bits tantos bytes como sea necesario, teniendo en
//cuenta que el byte final no debe ser desplazado.
if (i != 1){
B = B << (8*i);
}

//Se almacena en la variable final el resultado de la operación OR de
//ambos datos. Al realizar la operación OR con una variable que solo
//contiene ceros (variable Dato), el resultado es el valor de la otra
//variable. OR implica => 0+0=0, 0+1=1, 1+0=1, 1+1=1.
Dato = Dato | B;

//Se limpia la variable auxiliar para que en las siguientes iteraciones
//no se cambie el resultado de los otros bytes.
B = 0;
}
```

GUARDAR Y LEER DATOS EN UNA TARJETA MICROSD

El módulo de tarjetas microSD nos permite interactuar (leer y escribir) con tarjetas una memoria Micro SD que son las más comunes en el mercado.

Arduino tiene una librería para usar estas memorias, que funciona con cualquiera de los módulos antes mencionados. La librería ya viene junto con el IDE de arduinos, por lo que no necesitamos instalar ni descargar nada.

Para poder usar la librería en nuestro Sketch es necesario incluir a la librería SD al inicio del código:

```
#include <SD.h>
```

Si lo desea puede estudiar todas las funciones que dispone esta librería en la referencia oficial de arduino https://www.arduino.cc/en/Reference/SD

FUNCIONES BÁSICAS

SD.begin(cspin)

Inicializa la biblioteca SD y la tarjeta, como parámetro se le indica el pin CS al que está conectado el modulo, si no se especifica cspin , se usa el valor por defecto del CS por hardware. Los demás pines deben estar conectados al SPI por hardware del Arduino.

SD.exists(filename)

Comprueba si existe el archivo especificado, filename es el nombre del archivo y/o directorio en la tarjeta SD si este existe la función nos retorna un true, de lo contrario retorna false.

SD.mkdir(directory)

Crea el directorio especificado, si los subdirectorios no existen, también se crearan. Por ejemplo: SD.mkdir("Arduino/proyecto1/archivos), crea la carpeta "archivos" y si las carpetas Arduino y proyecto1 no existen, entonces también se crearan. La función retorna true si la creación del directorio fue exitosa de lo contrario nos retorna un false

SD.remove(filename)

Elimina el archivo (filename) de la tarjeta SD, se debe de incluir el directorio. Solo elimina el archivo más no el directorio. Devuelve true se logra eliminar el archivo de lo contrario nos retorna un false.

SD.rmdir(dirname)

Eliminar el directorio (dirname) de la tarjeta SD. El directorio debe estar vacío. Devuelve TRUE si la eliminación del directorio tuvo éxito o FALSE en caso contrario.

SD.open(filepath, mode)

Abre el archivo especificado y se debe de incluir el directorio si el archivo está en carpetas. Si el archivo no existe, se creara un archivo con el nombre especificado, pero no será posible crear el directorio si este no existe. Se puede abrir un archivo como solo lectura (si mode es FILE_READ) o como lectura y escritura (si mode es FILE_WRITE), el modo por defecto en caso no se especifique es FILE_READ

Ésta función nos retorna un objeto tipo FILE, el cual es necesario declararlo antes como una variable. Por ejemplo:

```
File myFile;

myFile = SD.open("archivo.txt", FILE_WRITE);
```

FUNCIONES DE LA CLASE FILE:

file.available()

Compruebe si hay bytes disponibles para leer en el archivo y retorna el número de bytes disponibles

file.read()

Lee un byte de la variable File (archivo abierto anteriormente con SD.open())

file.write(data)

Escribe un byte en el archivo, el archivo debe estar abierto en modo lectura y escritura. Usando file.write(buf, len) se puede escribir un array de byte (buf) pero se debe especificar el tamaño (len).

file.print(data)

Esta función tiene las mismas características que un Serial.print(); data puede ser una variable o texto, el cual será enviado como caracteres. Si queremos agregar al final un salto o nueva línea se usa file.println(data)

file.size()

Retorna el tamaño en bytes del archivo

file.position()

Retorna la posición actual en donde se leerá o escribirá el siguiente byte.

file.seek(pos)

Nos ubicamos en una posición específica en el archivo. Pos debe ser un número entre 0 y el tamaño en bytes del archivo

file.close()

Cerramos el archivo, y recién en este momento los datos se guardan en la SD, pudiendo extraer de forma segura nuestra SD.

CONECTAR CON ARDUINO

Este módulo utiliza la comunicación SPI por lo que utilizaremos los pines MOSI, MISO y SCK.

Módulo SD	Arduino Uno, Nano	Arduino Mega
CS	4	4
SCK	13	52
MOSI	11	51
MISO	12	50
VCC	5V	5V
GND	GND	GND

EJEMPLO 1: LEER UN ARCHIVO DE LA SD

En este ejemplo vamos a leer un archivo de texto desde la SD.

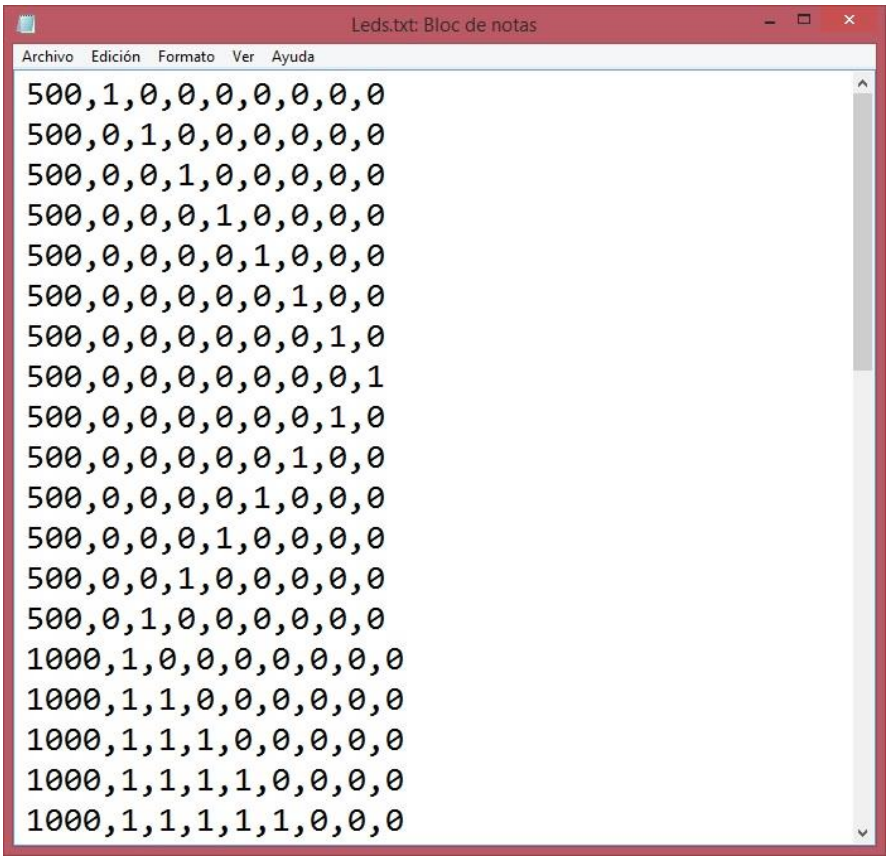
Para esto insertamos nuestra SD a la PC, abrimos y creamos un archivo de hoja de texto, por ejemplo archivo.txt, e ingresamos el texto que posteriormente vamos a leer desde Arduino

```
// DECLARACIÓN DE LIBRERIAS
#include <SD.h>

File myFile;
int cspin=4;

// INICIALIZACIÓN DEL PROGRAMA (SE EJECUTA SOLO UNA VEZ)
void setup()
{
  Serial.begin(9600);
  Serial.print("Iniciando SD ...");
  if (!SD.begin(cspin)) {
    Serial.println("No se pudo inicializar");
    return;
  }
  Serial.println("inicializacion exitosa");

  myFile = SD.open("archivo.txt");//abrimos  el archivo
  if (myFile) {
    Serial.println("archivo.txt:");
    while (myFile.available()) {
       Serial.write(myFile.read());
    }
    myFile.close(); //cerramos el archivo
  } else {
    Serial.println("Error al abrir el archivo");
  }
}

// LOOP CENTRAL DEL PROGRAMA (SE EJECUTA CONSTANTEMENTE DE FORMA SECUENCIAL
void loop()
{

}
```

EJEMPLO 2: LEER Y PROCESAR DATOS DESDE LA SD

En un archivo de texto vamos a guardar datos para encender y apagar leds y con esto ejecutar diferentes secuencias que estarán guardadas en la SD.

Conectamos ocho leds en los pines del 2 al 10, excluyendo el 4, con su resistencia respectiva.



El primer número de cada línea corresponde a la pausa o tiempo que va a estar presente la secuencia, los siguientes ocho números corresponde a la secuencia o estado de cada uno de los ocho leds.

```
#include <SD.h>

File myFile;
int cspin=4;
int UltimaPocicion=0;
int pausa=1000;
int PinLeds[8]={2,3,5,6,7,8,9,10};//Pines de los Leds
void setup()
{
  Serial.begin(9600);
  Serial.print("Iniciando SD ...");
  if (!SD.begin(cspin)) {
    Serial.println("No se pudo inicializar");
    return;
  }
  Serial.println("inicializacion exitosa");
  for(int i=0;i<8;i++)
  {
    pinMode(PinLeds[i],OUTPUT);
  }

}
```

```
void loop()
{
  myFile = SD.open("Leds.txt");//abrimos  el archivo
  int totalBytes=myFile.size();
  String cadena="";

  if (myFile) {
      if(UltimaPocicion>=totalBytes)   UltimaPocicion=0;
      myFile.seek(UltimaPocicion);

      //--Leemos una línea de la hoja de texto--------------
      while (myFile.available()) {

       char caracter=myFile.read();
          cadena=cadena+caracter;
          UltimaPocicion=myFile.position();
          if(caracter==10)//ASCII de nueva de línea
          {
            break;
          }
      }
      //--------------------------------------------------
      myFile.close(); //cerramos el archivo
      Serial.print("Cadena Leida:");
      Serial.print(cadena);
      //-----------procesamos la cadena------------
      int index=0;
      char c=cadena[index++];
      pausa=0;
      while (c >= '0' && c <= '9')
      {
        pausa = 10*pausa + (c - '0');
        c = cadena[index++];
      }
      Serial.print("pausa=");
      Serial.print(pausa);
      Serial.print("   LEDS |");
      for(int i=0;i<8;i++)
      {
        if(cadena[index+i*2]=='1')
        {
          digitalWrite(PinLeds[i], HIGH);
          Serial.print(" 1 |");

        }
        else
        {
          digitalWrite(PinLeds[i], LOW);
          Serial.print(" 0 |");
        }
      }
      Serial.println();
      Serial.println();

  } else {
    Serial.println("Error al abrir el archivo");
```

```
  }
  delay(pausa);
}
```

EJEMPLO 3: GUARDAR DATOS EN LA SD

En este ejemplo guardaremos información de variables y sensores en nuestra SD

```
#include <SD.h>
File myFile;
int cspin=4;

void setup()
{
  Serial.begin(9600);
  Serial.print("Iniciando SD ...");
  if (!SD.begin(cspin)) {
    Serial.println("No se pudo inicializar");
    return;
  }
  Serial.println("inicializacion exitosa");
}

void loop()
{
  myFile = SD.open("datalog.txt", FILE_WRITE);//abrimos  el archivo

  if (myFile) {
        Serial.print("Escribiendo SD: ");
        int sensor1 = analogRead(0);
        int sensor2 = analogRead(1);
        int sensor3 = analogRead(2);
        myFile.print("Tiempo(ms)=");
        myFile.print(millis());

        myFile.print(sensor1);
        myFile.print(", sensor2=");
        myFile.print(sensor2);
        myFile.print(", sensor3=");
        myFile.println(sensor3);

        myFile.close(); //cerramos el archivo

        Serial.print("Tiempo(ms)=");
        Serial.print(millis());
        Serial.print(", sensor1=");
        Serial.print(sensor1);
        Serial.print(", sensor2=");
        Serial.print(sensor2);
        Serial.print(", sensor3=");
        Serial.println(sensor3);

  } else {
    Serial.println("Error al abrir el archivo");
  }
  delay(100);
}
```

CONECTAR ARDUINO A LA RED LOCAL/INTERNET

El Módulo Ethernet ENC28J60 es popular por su costo, existen librerías para hacerlo trabajar con Arduino, y aunque consume más memoria que el Ethernet Shield para aplicaciones básicas es muy útil.

En internet puedes encontrar gran cantidad de información y ejemplos de este módulo en la red, así como diferentes librerías. La mejor y más documentada librería que hemos encontrado es la que utilizaremos en este ejemplo *Ethercard.*

Para conocer más de esta librería:

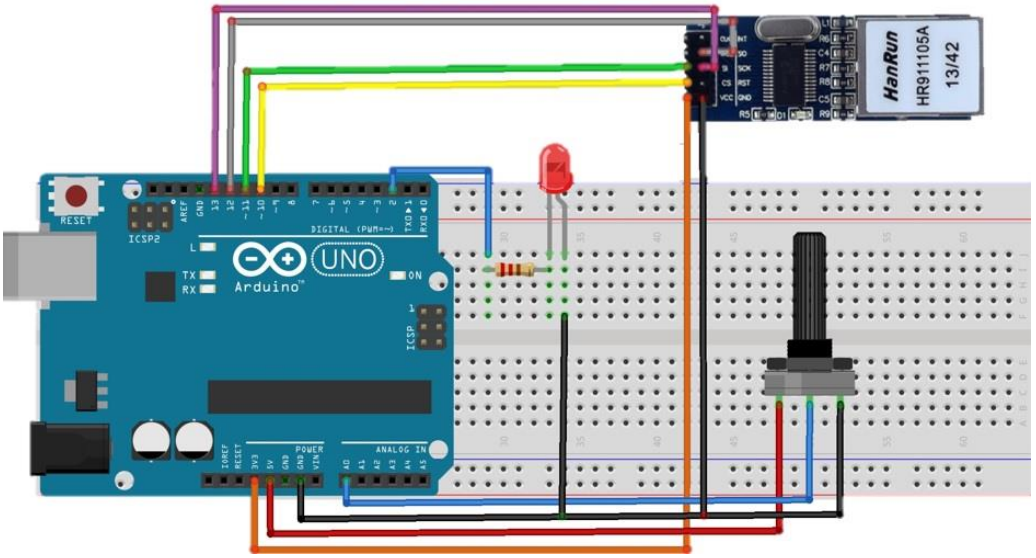
http://jeelabs.org/pub/docs/ethercard/index.html

CONECTAR CON ARDUINO

Este módulo se comunica con el arduino a través del protocolo de comunicaciones SPI, por lo que utilizaremos los siguientes pines de conexión:

Módulo ENC28J60	Arduino Uno
VCC	Pin 3.3V
GND	Pin GND
SS (CS)	Pin 10
MOSI (SI)	Pin 11
MISO (SO)	Pin 12
SCK	Pin 13

En esta práctica vamos a hacer lo siguiente, conectaremos un led y un potenciómetro y desde una página web alojada en el arduino, encenderemos y apagaremos el led y mostraremos el valor del potenciómetro.



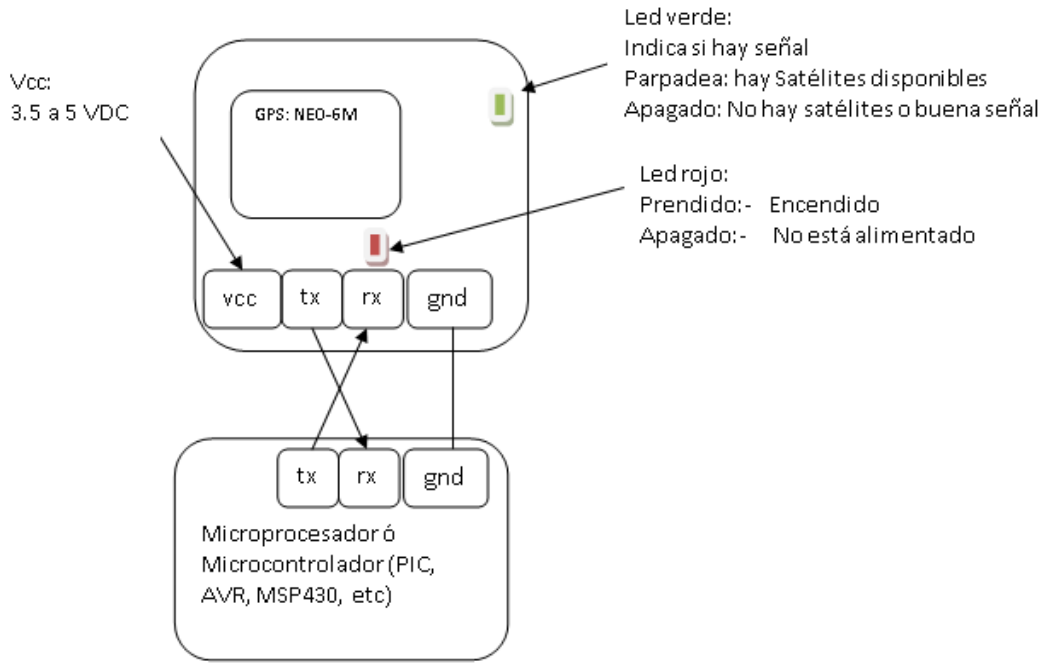
PROGRAMA DE EJEMPLO

```
#include <EtherCard.h>

static byte mymac[] = {0xDD,0xDD,0xDD,0x00,0x01,0x05};
static byte myip[] = {192,168,1,177};
byte Ethernet::buffer[700];

const int ledPin = 2;

char* EstadoLed="OFF";

void setup () {

  Serial.begin(9600);
  Serial.println("Test del Modulo  ENC28J60");

  if (!ether.begin(sizeof Ethernet::buffer, mymac, 10))
    Serial.println( "No se ha podido acceder a la controlador Ethernet");
 else
   Serial.println("Controlador Ethernet inicializado");

  if (!ether.staticSetup(myip))
    Serial.println("No se pudo establecer la dirección IP");

  Serial.println();

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}

static word homePage() {

 BufferFiller bfill = ether.tcpOffset();
 bfill.emit_p(PSTR("<!DOCTYPE HTML PUBLIC \r\n \"-//W3C//DTD HTML 4.01//EN\"
\"http://www.w3.org/TR/html4/strict.dtd \r\n\">"
     "Pragma: no-cache\r\n"
```

```
      "\r\n"
      "<meta http-equiv='refresh' content='1'/>"
      "<html><head><meta charset=\"UTF-8\"><title>Inven Technology
Experts</title></head>"
      "<body>"
      "<div style='text-align:center;'>"
      "<h1>Test del Módulo  ENC28J60</h1>"
      "Tiempo transcurrido : $L segundos"
      "<br /><br />Estado del LED: $S<br />"
      "<a href=\"/?status=ON\"><input type=\"button\" value=\"ON\"></a>"
      "<a href=\"/?status=OFF\"><input type=\"button\" value=\"OFF\"></a>"
      "<br /><br />Potenciómetro: $D (resolución de 1024)"
      "<br /><br />"
      "<a href='http://www.inven.es/'>www.inven.es</a>"
      "</body></html>"
      ),millis()/1000,EstadoLed,analogRead(0));

  return bfill.position();
}

void loop() {

  word len = ether.packetReceive();
  word pos = ether.packetLoop(len);

  if(pos) {
    // Comprobamos si se recibe status ON por método GET
    if(strstr((char *)Ethernet::buffer + pos, "GET /?status=ON") != 0) {
      Serial.println("Comando ON recibido");
      digitalWrite(ledPin, HIGH);
      EstadoLed = "ON";
    }

    if(strstr((char *)Ethernet::buffer + pos, "GET /?status=OFF") != 0) {
      Serial.println("Comando OFF recibido");
      digitalWrite(ledPin, LOW);
       EstadoLed= "OFF";
    }
    ether.httpServerReply(homePage()); // se envia página Web
  }
}
```

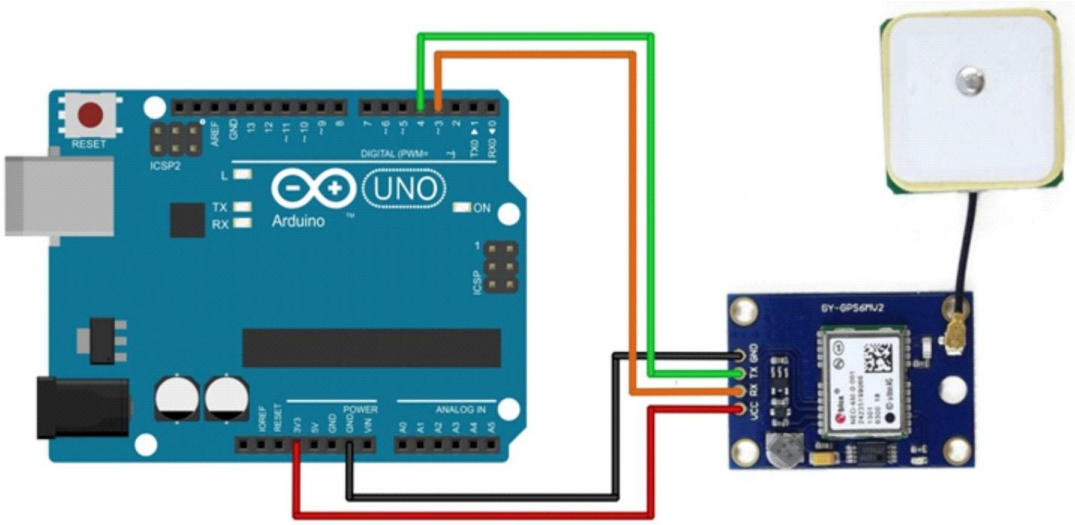
MÓDULO GPS NEO MV2

El módulo GPS en su modelo GY-GPS6MV2 viene con un módulo de serie U-Blox NEO 6M equipado en el PCB, una EEPROM con configuración de fábrica, una pila de botón para mantener los datos de configuración en la memoria EEPROM, un indicador LED y una antena cerámica. También posee los pines o conectores Vcc, Rx, Tx y Gnd por el que se puede conectar a algún microcontrolador mediante una interfaz serial. Para que nuestro módulo GPS funcione a la perfección se recomienda hacer las pruebas en un ambiente abierto o cercano a la ventana para una correcta recepción de la señal.

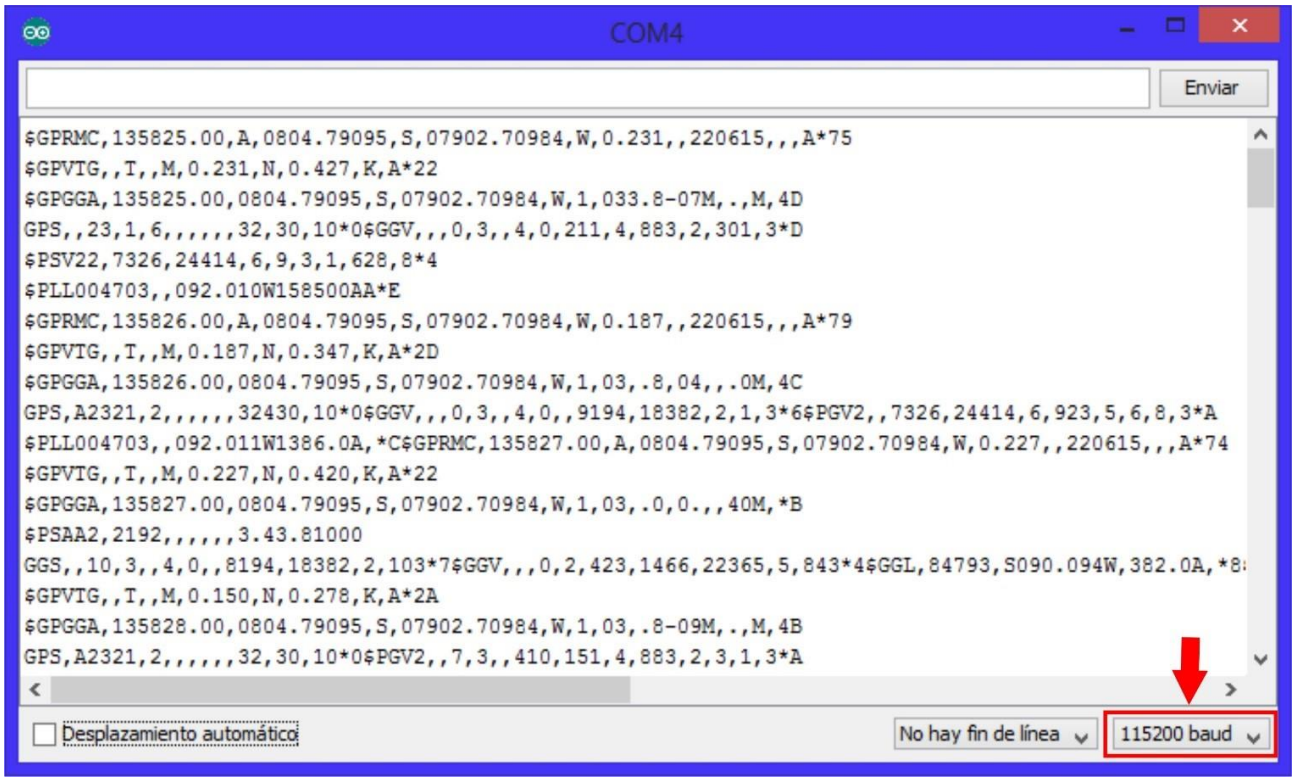


COMO SE CONECTA A ARDUINO

La comunicación del módulo es a través del puerto serie. Como los pines 0 y 1 de comunicación serie por defecto les tendremos ocupados con la comunicación serie con el PC, conectaremos el módulo GPS a los pines 3 y 4 emulando el puerto serie por software con la librería SoftwareSerial.h que viene incluida en el IDE de Arduino.



Los datos que recibimos del módulo GPS siguen el protocolo NMEA (siglas de National Marine Electronics Asociation), las cuales son sentencias estándares para la recepción de datos GPS. Una de ellas y la más usada son las sentencias $GPRMC, las cuales tienen la siguiente estructura:



$GPRMC,044235.000,A,4322.0289,N,00824.5210,W,0.39,65.46,020615,,,A*44

Donde si analizamos la trama de este ejemplo y basándose en el protocolo NMEA, podríamos determinar las siguientes variables:

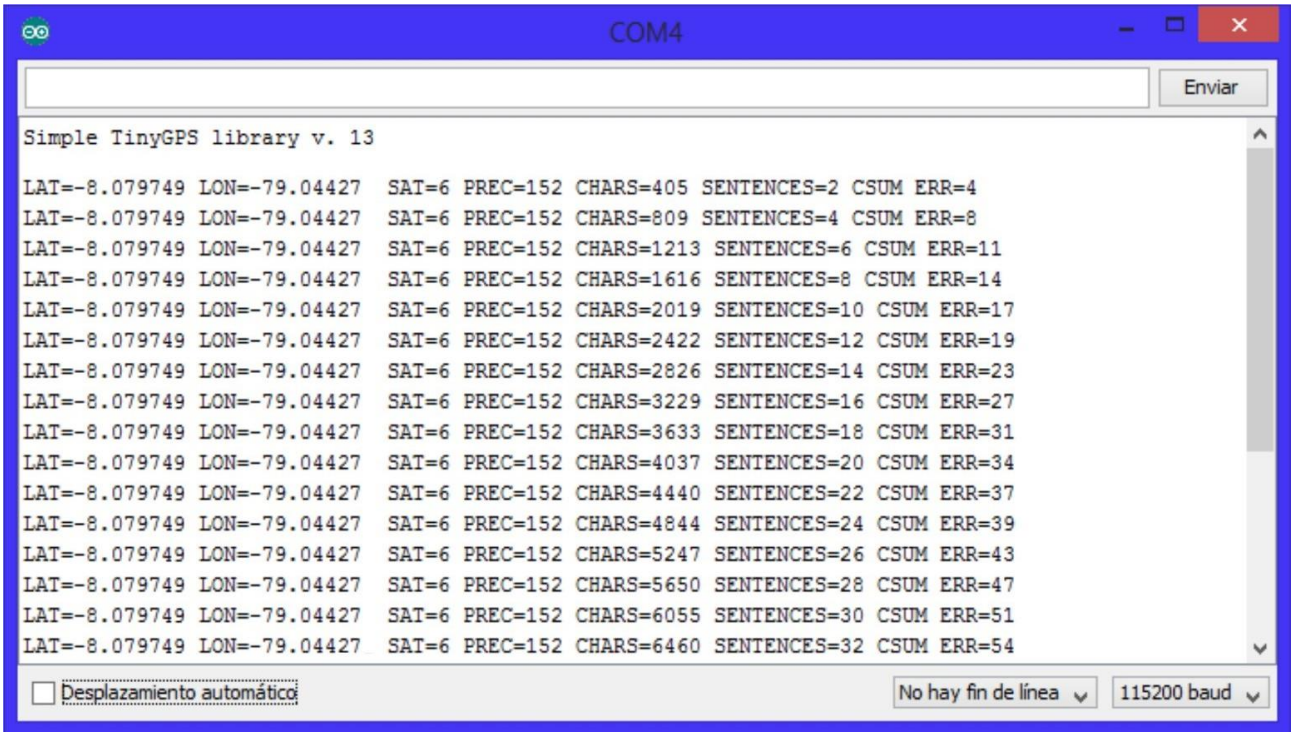
- **044235.000** representa la hora GMT (04:42:35)
- **"A"** es la indicación de que el dato de posición está fijado y es correcto. "V" sería no válido
- **4322.0289** representa la longitud (43º 22.0289´)
- **N** representa el Norte
- **00824.5210** representa la latitud (8º 24.5210´)
- **W** representa el Oeste
- **0.39** representa la velocidad en nudos
- **65.46** representa la orientación en grados
- **020615** representa la fecha (2 de Junio del 2015)

Como vimos, de la trama de datos que nos envía nuestro módulo GPS podemos obtener varias variables, siendo las importantes para proyectos de posicionamiento la latitud y la longitud. Para ello, vamos a hacer uso de la librería **TinyGPS**.

Cargamos el ejemplo **Simple_test** y nos aseguramos de ajustar la velocidad de comunicación serial con el módulo gps a 9600

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>

/* This sample code demonstrates the normal use of a TinyGPS object.
   It requires the use of SoftwareSerial, and assumes that you have a
   9600-baud serial GPS device hooked up on pins 4(rx) and 3(tx).
*/

TinyGPS gps;
SoftwareSerial ss(4, 3);

void setup()
{
  Serial.begin(115200);
  ss.begin(9600);

  Serial.print("Simple TinyGPS library v. ");
Serial.println(TinyGPS::library_version());
  Serial.println();
}

void loop()
{
  bool newData = false;
  unsigned long chars;
  unsigned short sentences, failed;
```

```
  // For one second we parse GPS data and report some key values
  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (ss.available())
    {
      char c = ss.read();
      // Serial.write(c); // uncomment this line if you want to see the GPS data
flowing
      if (gps.encode(c)) // Did a new valid sentence come in?
        newData = true;
    }
  }

  if (newData)
  {
    float flat, flon;
    unsigned long age;
    gps.f_get_position(&flat, &flon, &age);
    Serial.print("LAT=");
    Serial.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
    Serial.print(" LON=");
    Serial.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
    Serial.print(" SAT=");
    Serial.print(gps.satellites() == TinyGPS::GPS_INVALID_SATELLITES ? 0 :
gps.satellites());
    Serial.print(" PREC=");
    Serial.print(gps.hdop() == TinyGPS::GPS_INVALID_HDOP ? 0 : gps.hdop());
  }

  gps.stats(&chars, &sentences, &failed);
  Serial.print(" CHARS=");
  Serial.print(chars);
  Serial.print(" SENTENCES=");
  Serial.print(sentences);
  Serial.print(" CSUM ERR=");
  Serial.println(failed);
  if (chars == 0)
    Serial.println("** No characters received from GPS: check wiring **");
}
```


COM4
Enviar
Simple TinyGPS library v. 13
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=405 SENTENCES=2 CSUM ERR=4
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=809 SENTENCES=4 CSUM ERR=8
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=1213 SENTENCES=6 CSUM ERR=11
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=1616 SENTENCES=8 CSUM ERR=14
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=2019 SENTENCES=10 CSUM ERR=17
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=2422 SENTENCES=12 CSUM ERR=19
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=2826 SENTENCES=14 CSUM ERR=23
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=3229 SENTENCES=16 CSUM ERR=27
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=3633 SENTENCES=18 CSUM ERR=31
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=4037 SENTENCES=20 CSUM ERR=34
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=4440 SENTENCES=22 CSUM ERR=37
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=4844 SENTENCES=24 CSUM ERR=39
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=5247 SENTENCES=26 CSUM ERR=43
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=5650 SENTENCES=28 CSUM ERR=47
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=6055 SENTENCES=30 CSUM ERR=51
LAT=-8.079749 LON=-79.04427 SAT=6 PREC=152 CHARS=6460 SENTENCES=32 CSUM ERR=54
Desplazamiento automático
No hay fin de línea
115200 baud

COMUNICACIONES ARDUINO CON BLUETOOTH

Antes de empezar es necesario saber que existen diferentes modelos de módulos Bluetooth entre los más populares se encuentran HC-06 y el HC-05. Los módulos Bluetooth HC-06 se comportan solo como esclavo, esperando peticiones de conexión, Si algún dispositivo se conecta, el HC-06 transmite a este todos los datos que recibe del Arduino y viceversa. Los módulos Bluetooth HC-05 se pueden compartar como Mater y Esclavo, esperando peticiones de conexión o iniciando la comunicación.

DIFERENCIAS ENTRE HC-05 Y HC-06

Los módulos HC-06 y HC-05 son prácticamente idénticos a simple vista. Físicamente se diferencian por el número de pines. En el HC-06 tiene un conector de 4 pines mientras que el HC-05 trae uno de 6 pines

CONECTAR EL HC-05 COMO ESCLAVO

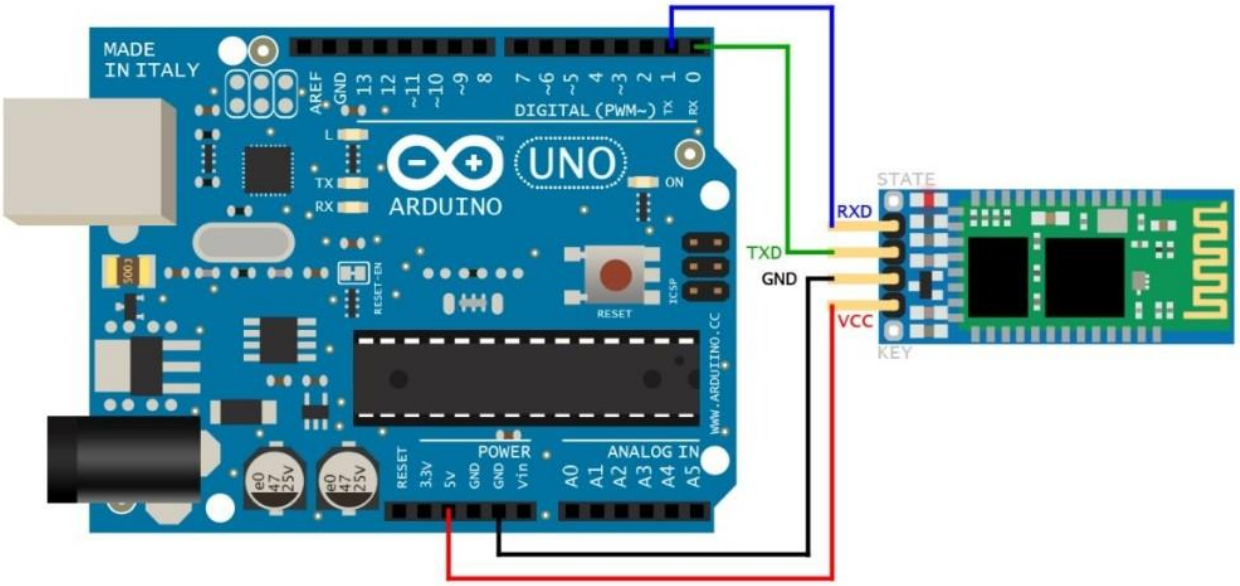
Para conectar el Módulo bluetooth como esclavo, se utilizan solo 4 pines.

- **Vcc**, Voltaje positivo de alimentación, aquí hay tener cuidado porque hay módulos que solo soportan voltajes de 3.3V, pero en su mayoría ya vienen acondicionados para q trabajen en el rango de 3.3V a 6V pero es bueno revisar los dato técnicos de nuestro modulo antes de hacer las conexiones

- **GND**, Voltaje negativo de alimentación, se tienen que conectar al GND del Arduino o al GND de la placa que se esté usando.

- **TX,** Pin de Transmisión de datos, por este pin el HC-05 transmite los datos que le llegan desde la PC o Móvil mediante bluetooth, este pin debe ir conectado al pin RX del Arduino

- **RX**, pin de Recepción, a través de este pin el HC-05 recibirá los datos del Arduino los cuales se transmitirán por Bluetooth, este pin va conectado al Pin TX del Arduino



IMPORTANTE

Para cargar el programa al Arduino, desconectaremos los pines RX0 y TX0 del Arduino, pues internamente el Arduino trabaja con los mismos pines para cargar el programa y si están conectados al módulo Bluetooth, no nos dejara cargar (para evitar este inconveniente se puede usar el software serial y usar otros pines).

El módulo HC-05 viene por defecto configurado de la siguiente forma:

- Modo o role: Esclavo
- Nombre por defeco: HC-05
- Código de emparejamiento por defecto: 1234
- La velocidad por defecto (baud rate): 9600

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available())
  {
    char dato=Serial.read();
    Serial.print("Dato recibido: ");
    Serial.println(dato);
  }
}
```

Este sencillo programa devuelve al serial cualquier dato que se haya recibido por el bluetoth. Para comunicar el ordenador con el bluetooth podemos usar el software putty, y para comunicar con el móvil podemos utilizar la aplicación BlueTerm que se puede descargar desde el play Store

ARDUINO BLUETOOTH Y APLICACIONES MÓVILES

Se pueden crear aplicaciones móviles con las que controlar tu arduino a través del Bluetooth, pero en esta ocasión usaremos una aplicación que actualmente está en el mercado denominada ArduDroid.

http://www.techbitar.com/ardudroid-simple-bluetooth-control-for-arduino-and-android.html

ARDUDROID es una sencilla aplicación para Android para ayudarle a controlar los pines de la placa Arduino Uno desde el teléfono Android de forma inalámbrica. ArduDroid emplea una sencilla interfaz de Android de usuario para: 1) controlar los pines digitales y PWM del Arduino Uno 2) envía comandos de texto para Arduino 3) y recibir datos de Arduino a través del puerto serie.

Puede utilizar ArduDroid para enviar comandos a Arduino y controlar un relé para encender aparatos eléctricos de encendido / apagado, controlar un servo robot, velocidad de aumento / disminución de un motor y el sentido inverso, regular la luz, acelerar un ventilador, y así sucesivamente.

```
/*
 PROJECT: ArduDroid
 PROGRAMMER: Hazim Bitar (techbitar at gmail dot com)
 DATE: Oct 31, 2013
 FILE: ardudroid.ino
 LICENSE: Public domain
*/

#define START_CMD_CHAR '*'
#define END_CMD_CHAR '#'
#define DIV_CMD_CHAR '|'
#define CMD_DIGITALWRITE 10
#define CMD_ANALOGWRITE 11
#define CMD_TEXT 12
#define CMD_READ_ARDUDROID 13
#define MAX_COMMAND 20  // max command number code. used for error checking.
#define MIN_COMMAND 10  // minimum command number code. used for error checking.
#define IN_STRING_LENGHT 40
```

```
#define MAX_ANALOGWRITE 255
#define PIN_HIGH 3
#define PIN_LOW 2

String inText;

void setup() {
  Serial.begin(9600);
  Serial.println("ArduDroid 0.12 Alpha by TechBitar (2013)");
  Serial.flush();
}

void loop()
{
  Serial.flush();
  int ard_command = 0;
  int pin_num = 0;
  int pin_value = 0;

  char get_char = ' ';  //read serial

  // wait for incoming data
  if (Serial.available() < 1) return; // if serial empty, return to loop().

  // parse incoming command start flag
  get_char = Serial.read();
  if (get_char != START_CMD_CHAR) return; // if no command start flag, return to
loop().

  // parse incoming command type
  ard_command = Serial.parseInt(); // read the command

  // parse incoming pin# and value
  pin_num = Serial.parseInt(); // read the pin
  pin_value = Serial.parseInt();  // read the value

  // 1) GET TEXT COMMAND FROM ARDUDROID
  if (ard_command == CMD_TEXT){
    inText =""; //clears variable for new input
    while (Serial.available())  {
      char c = Serial.read();  //gets one byte from serial buffer
      delay(5);
      if (c == END_CMD_CHAR) { // if we the complete string has been read
        // add your code here
        Serial.println(inText);
        break;
      }
      else {
        if (c !=  DIV_CMD_CHAR) {
          inText += c;
          delay(5);
        }
      }
    }
  }
```

```
  // 2) GET digitalWrite DATA FROM ARDUDROID
  if (ard_command == CMD_DIGITALWRITE){
    if (pin_value == PIN_LOW) pin_value = LOW;
    else if (pin_value == PIN_HIGH) pin_value = HIGH;
    else return; // error in pin value. return.
    set_digitalwrite( pin_num,  pin_value);  // Uncomment this function if you wish to
use
    return;  // return from start of loop()
  }

  // 3) GET analogWrite DATA FROM ARDUDROID
  if (ard_command == CMD_ANALOGWRITE) {
    analogWrite(  pin_num, pin_value );
    // add your code here
    return;  // Done. return to loop();
  }

  // 4) SEND DATA TO ARDUDROID
  if (ard_command == CMD_READ_ARDUDROID) {
    // char send_to_android[] = "Place your text here." ;
    // Serial.println(send_to_android);   // Example: Sending text
    Serial.print(" Analog 0 = ");
    Serial.println(analogRead(A0));  // Example: Read and send Analog pin value to
Arduino
    return;  // Done. return to loop();
  }
}

// 2a) select the requested pin# for DigitalWrite action
void set_digitalwrite(int pin_num, int pin_value)
{
  switch (pin_num) {
  case 13:
    pinMode(13, OUTPUT);
    digitalWrite(13, pin_value);
    // add your code here
    break;
  case 12:
    pinMode(12, OUTPUT);
    digitalWrite(12, pin_value);
    // add your code here
    break;
  case 11:
    pinMode(11, OUTPUT);
    digitalWrite(11, pin_value);
    // add your code here
    break;
  case 10:
    pinMode(10, OUTPUT);
    digitalWrite(10, pin_value);
    // add your code here
    break;
  case 9:
    pinMode(9, OUTPUT);
    digitalWrite(9, pin_value);
    // add your code here
```

```
    break;
  case 8:
    pinMode(8, OUTPUT);
    digitalWrite(8, pin_value);
    // add your code here
    break;
  case 7:
    pinMode(7, OUTPUT);
    digitalWrite(7, pin_value);
    // add your code here
    break;
  case 6:
    pinMode(6, OUTPUT);
    digitalWrite(6, pin_value);
    // add your code here
    break;
  case 5:
    pinMode(5, OUTPUT);
    digitalWrite(5, pin_value);
    // add your code here
    break;
  case 4:
    pinMode(4, OUTPUT);
    digitalWrite(4, pin_value);
    // add your code here
    break;
  case 3:
    pinMode(3, OUTPUT);
    digitalWrite(3, pin_value);
    // add your code here
    break;
  case 2:
    pinMode(2, OUTPUT);
    digitalWrite(2, pin_value);
    // add your code here
    break;
    // default:
    // if nothing else matches, do the default
    // default is optional
  }
}
```

ANEXO 1: COMO DESTRUIR TU ARDUINO

Golpearla con un martillo, disparar, tirarla a la piscina... No es a lo que nos referimos, vamos a mostrar cómo destruir eléctricamente una placa de Arduino. Mucha gente lo descubre "por las malas", un olor raro, un componente con marcas de quemadura, o el temido mensaje de "programador no sincronizado" suelen ser señales de que hemos aprendido la lección de la peor forma posible.

FORMA 1: CORTOCIRCUITAR UN PIN DE ENTRADA/SALIDA CON TIERRA

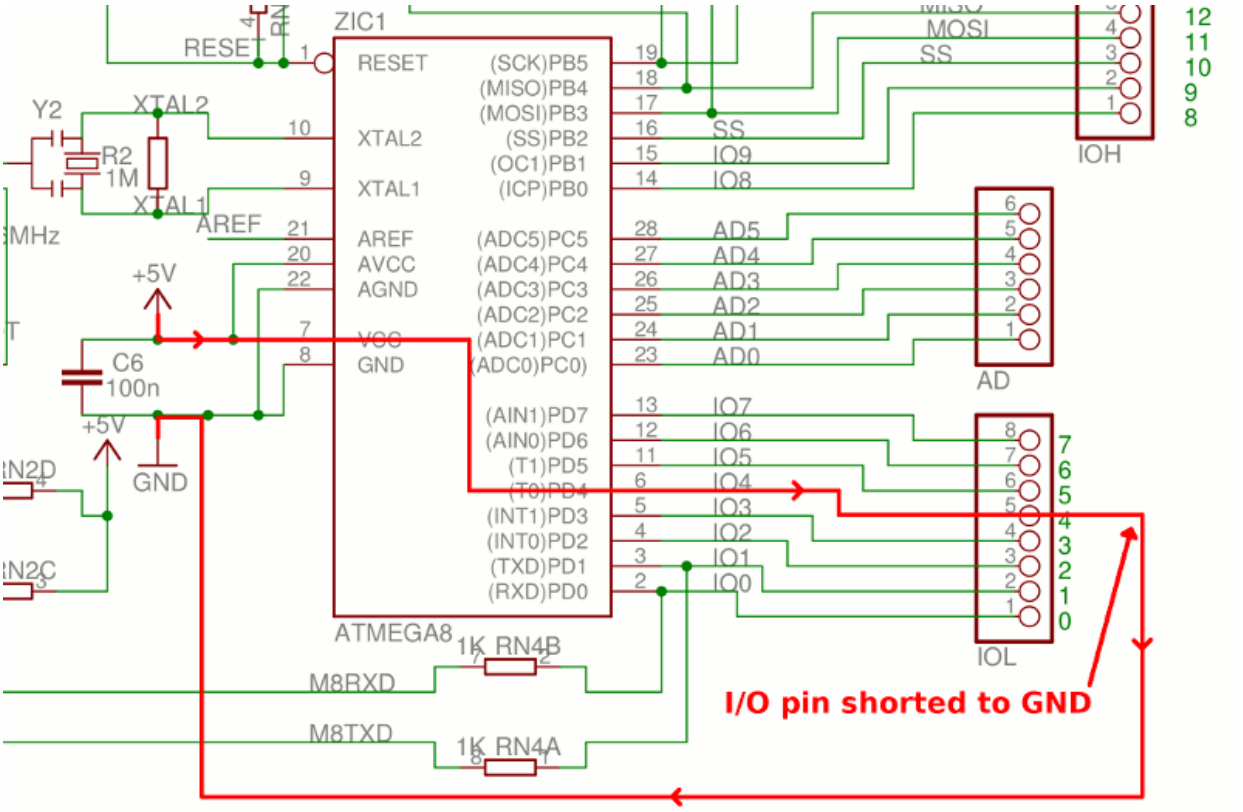
¿CÓMO OCURRE?

Configuramos uno de los pines cómo salida, lo establecemos en alto y este está directamente conectado a tierra, sin ningún componente que cree resistencia entre estos dos puntos (el pin de salida y el pin GND). El Pin de entrada/salida se sobrecarga y se destruye.

¿POR QUÉ?

Según las especificaciones de Arduino, la tensión máxima que puede entregar cada pin es de 40mA, pero sin nada que haga de resistencia entre el pin y tierra este entregará unos 200mA, tensión suficiente para destruir el microcontrolador.

Veamos el camino que toma la corriente cuando esto pasa:



FORMA 2: DOS PINES QUE SE CORTOCIRCUITAN MUTUAMENTE.

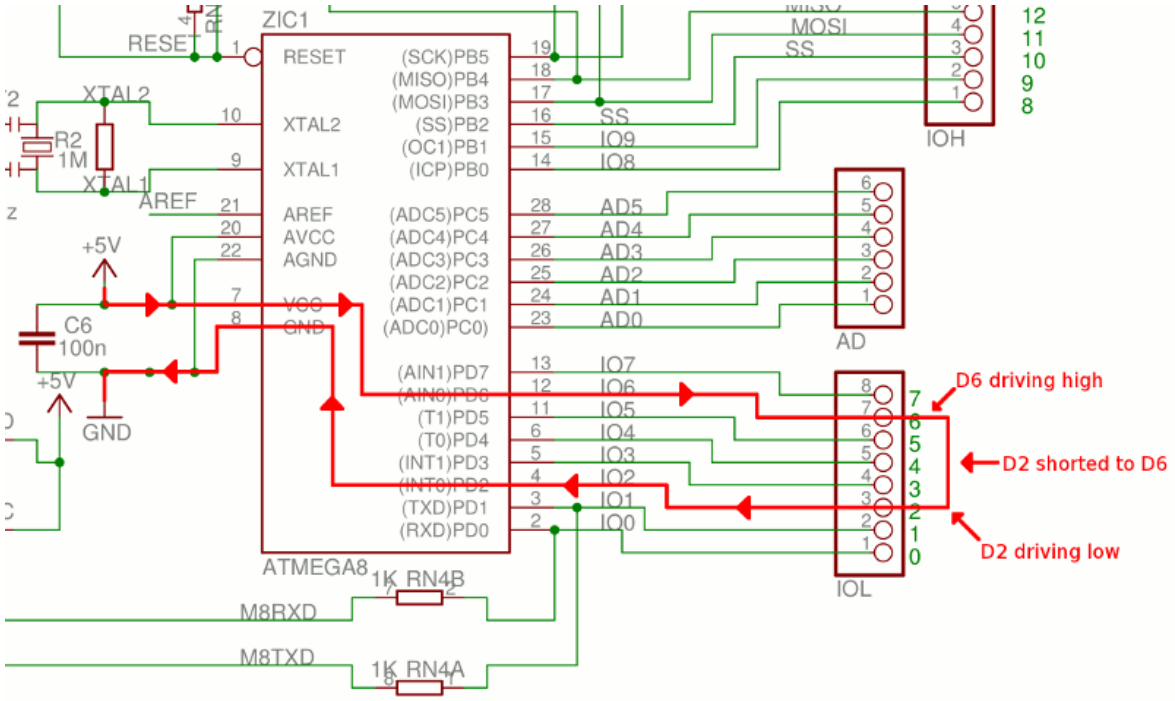
¿CÓMO OCURRE?

Configuramos ambos pines cómo entradas, uno tiene un valor alto, el otro un valor bajo. Ambos son conectados uno con otro sin que exista ningún componente entre ellos. En este caso, ambos pines se sobrecargan mutuamente y son destruidos.

¿POR QUÉ?

Por un motivo similar al de la forma 1. El pin que esta en baja en este caso haría las veces de tierra, y entregarían tensiones muy por encima del máximo del microcontrolador, lo que lo destruye.

Podemos ver el camino que realiza la corriente eléctrica en el siguiente esquema y veremos que es muy similar al motivo primero.



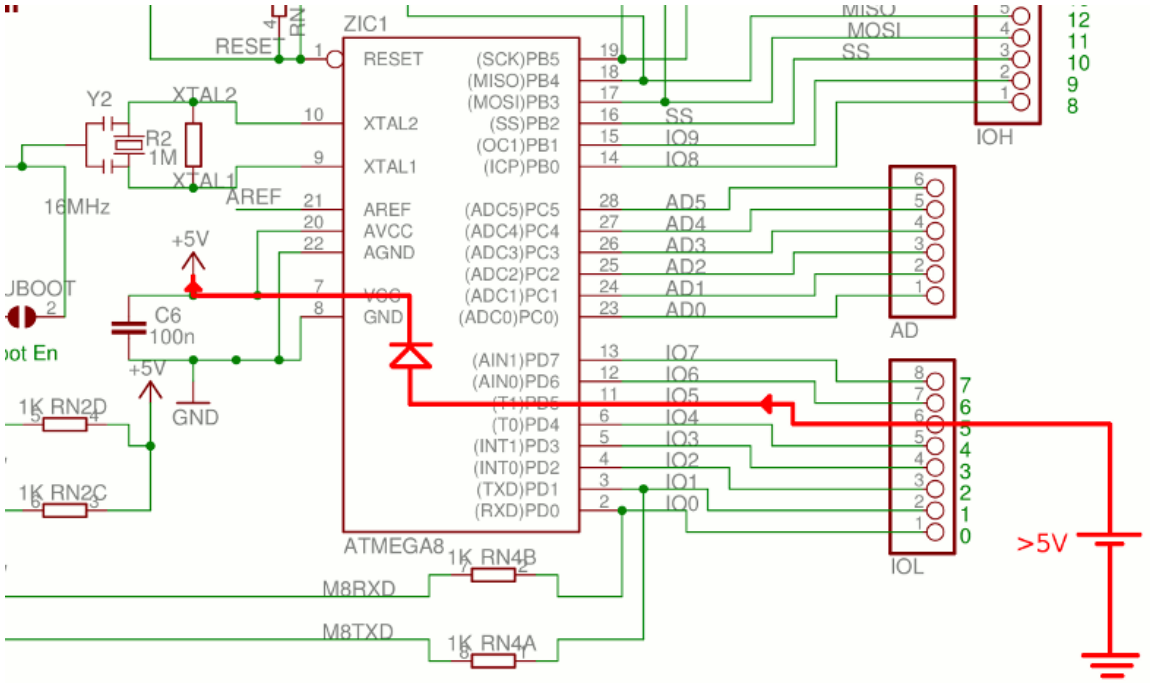
FORMA 3: SOBRE TENSIÓN EN LOS PINES DE ENTRADA/SALIDA

¿CÓMO OCURRE?

El voltaje máximo recomendado para cualquier pin de entrada/salida es de 5V (Salvo las versiones que operan a 3.3V), si aplicamos cualquier tensión por encima de 5.5V destruiremos nuestro Arduino.

¿POR QUÉ?

Los pines de Arduino cuentan con un diodo que protege el microcontrolador cuando se recibe una tensión superior a 5V, pero esta proyección esta pensada para descargas cortas, producidas por ejemplo por una descarga electrostática o un pequeño pico de tensión al arrancar, si esta sobre tensión dura, más que eso el diodo se quema, deja de proteger el pin del microcontrolador y este se quema.



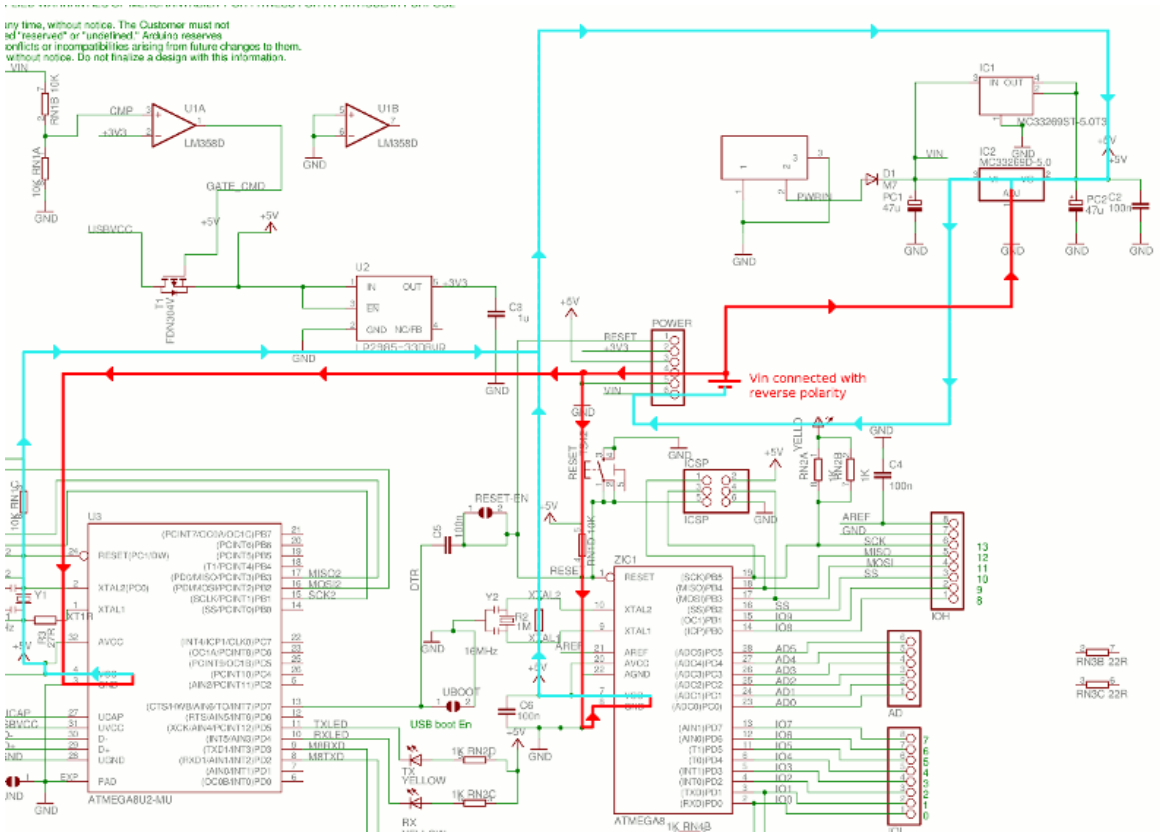
FORMA 4: INVERTIR LA POLARIDAD AL ALIMENTAR ARDUINO POR SU PATILLA VIN

¿CÓMO OCURRE?

Cuando queremos alimentar nuestro Arduino mediante su patilla Vin y por error conectamos Vin a tierra y GND a 5V.

¿POR QUÉ?

Arduino no cuenta con protección en caso de invertir su polaridad de alimentación, por lo que la corriente avanzará "en dirección contraria" por todo el circuito de Arduino, destruyendo el regulador de voltaje y probablemente el microcontrolador.



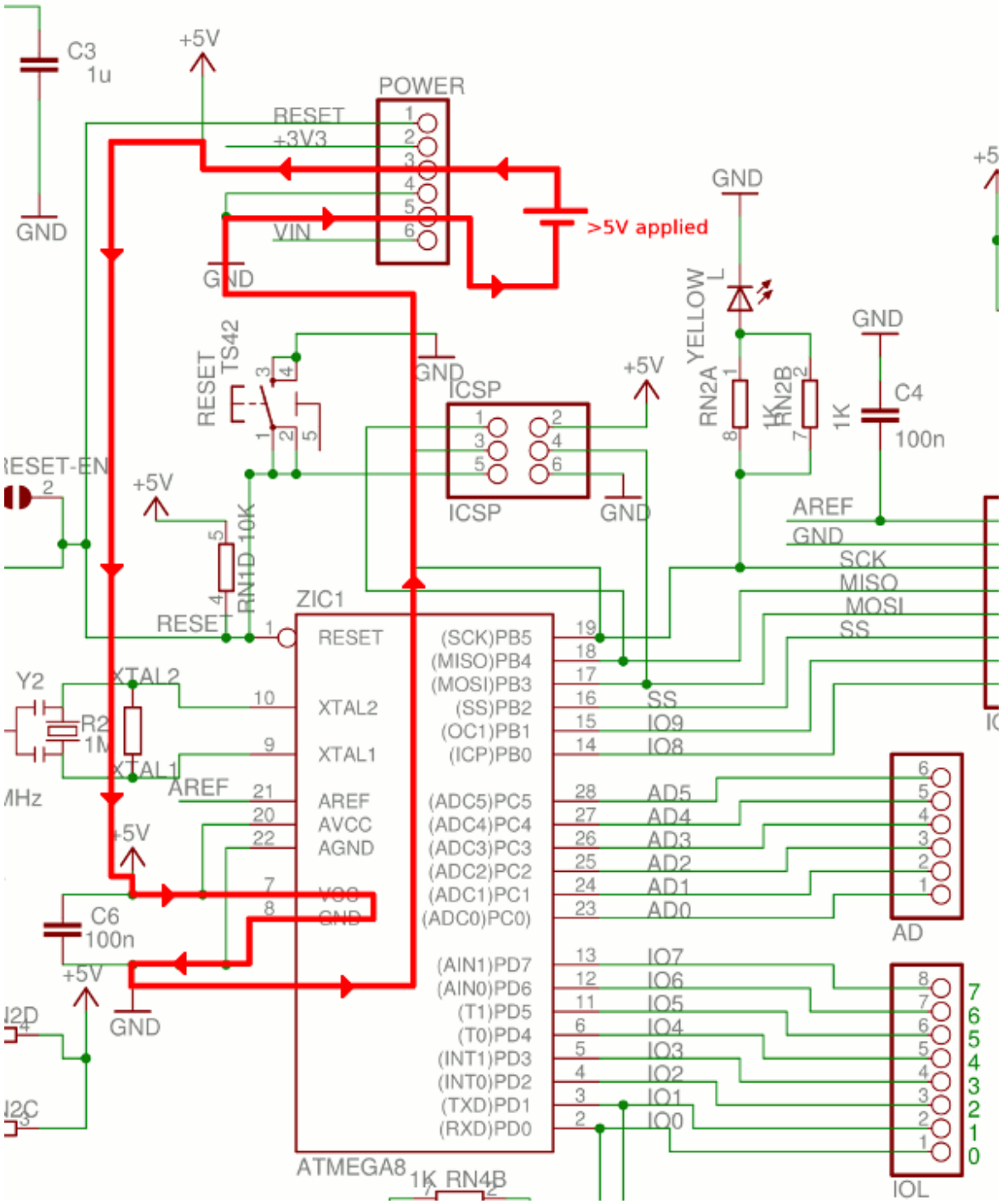
FORMA 5: APLICAR MÁS DE 5V A LA ALIMENTACIÓN DE ARDUINO.

¿CÓMO OCURRE?

Al aplicar 6V o más al pin de 5V de Arduino se queman muchos de los componentes de esta placa, incluso podría llegar la corriente al puerto USB del ordenador si este está conectado.

¿POR QUÉ?

No hay ningún tipo de protección ni regulador de voltaje en la pata 5V de Arduino, por lo que la electricidad correrá libremente por todos los componentes destruyéndolos a su paso.



FORMA 6: APLICAR MÁS DE 3.3V EN EL PIN DE 3.3V DE ARDUINO

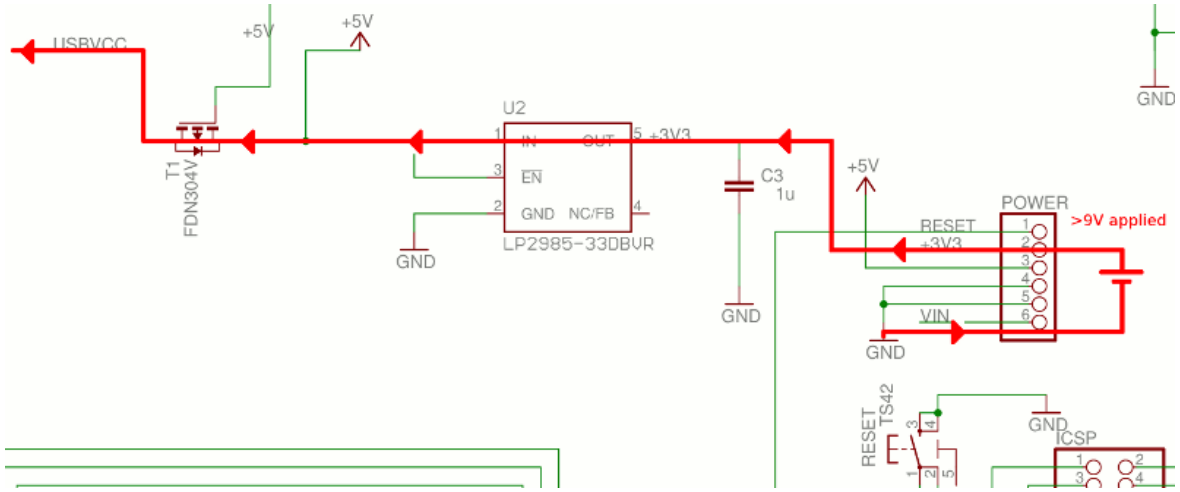
¿CÓMO OCURRE?

Si aplicamos más de 3.6V en la patilla de 3.3V podría dañar varios componentes de Arduino y probablemente también cualquier Shield que este conectado en ese momento. Si llegasen a entrar más de 9V se detruiria también el regulador de 3.3V y la

tensión podría causar daños en el puerto USB del ordenador si estuviese conectado en ese momento.

¿POR QUÉ?

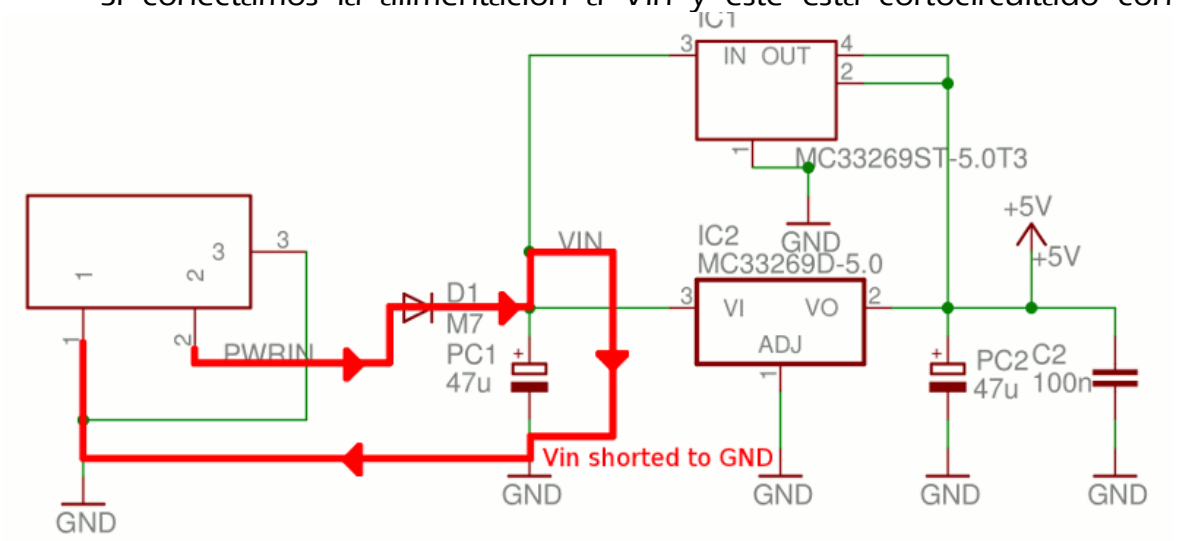
El pin de 3.3V no tiene circuitos de protección. La corriente entraría directamente al regulador de 3.3V y a cualquier componente conectado a esta tensión (Shield, módulos de bluetooth..etc). Si la tensión fuese de 9V o más el regulador sería destruido y la corriente llegaría también a los componentes alimentados normalmente por 5V



FORMA 7: CORTOCIRCUITAR VIN Y GND

¿CÓMO OCURRE?

Si conectamos la alimentación a Vin y este esta cortocircuitado con GND .



¿POR QUÉ?

Si esto ocurre, el diodo de protección de Arduino sería destruido, la tensión aumentaría de forma brusca y el calor disipado sería suficiente para dañar las pistas.

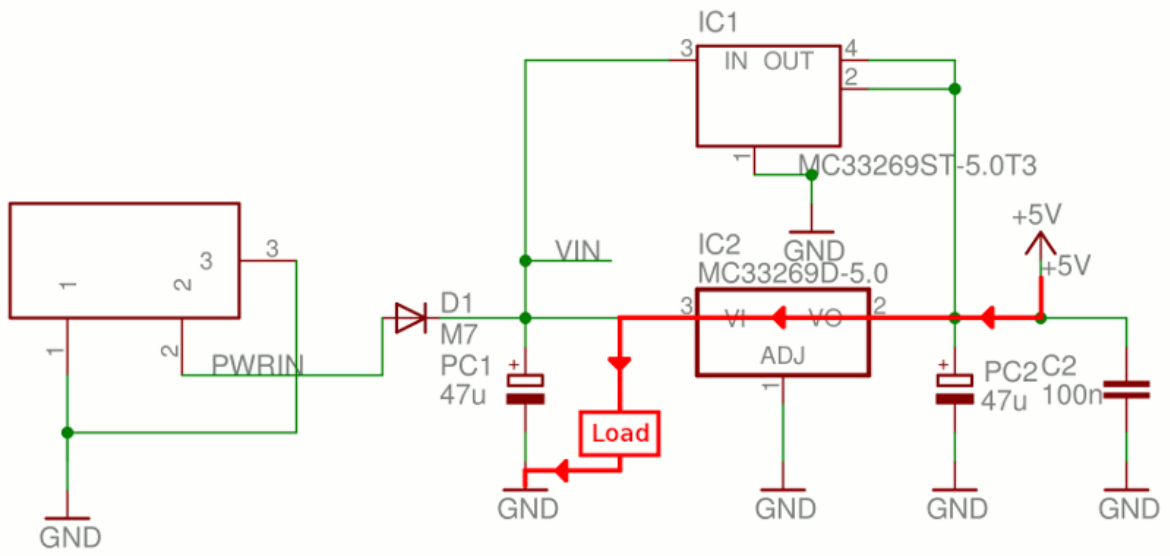
FORMA 8: UTILIZAR VIN CÓMO SALIDA DE CORRIENTE.

¿CÓMO OCURRE?

Al alimentar la placa mediante la patilla 5V y utilizar Vin para alimentar componentes.

¿POR QUÉ?

El regulador no tiene protección para evitar que la corriente circule a la inversa. Al realizar esta conexión estamos haciendo que la corriente circule hacia atrás, lo que destruirá el regulador.



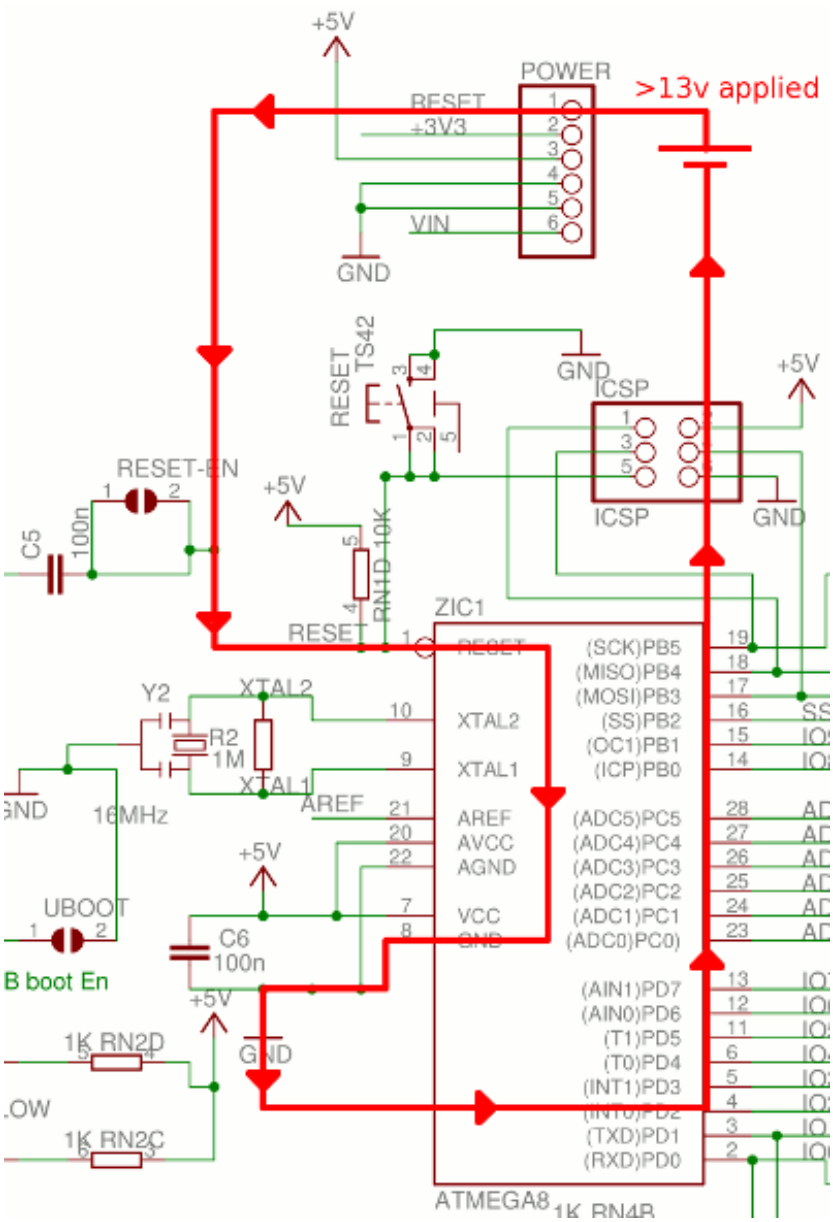
FORMA 9: APLICAR MÁS DE 13V EN EL RESET.

¿CÓMO OCURRE?

Si aplicamos más de 13 en la patilla de reset de Arduino dañaremos el microcontrolador.

¿POR QUÉ?

13V es el máximo que soporta la patilla de reset de microcontrolador de Arduino. Un voltaje superior daña el chip.



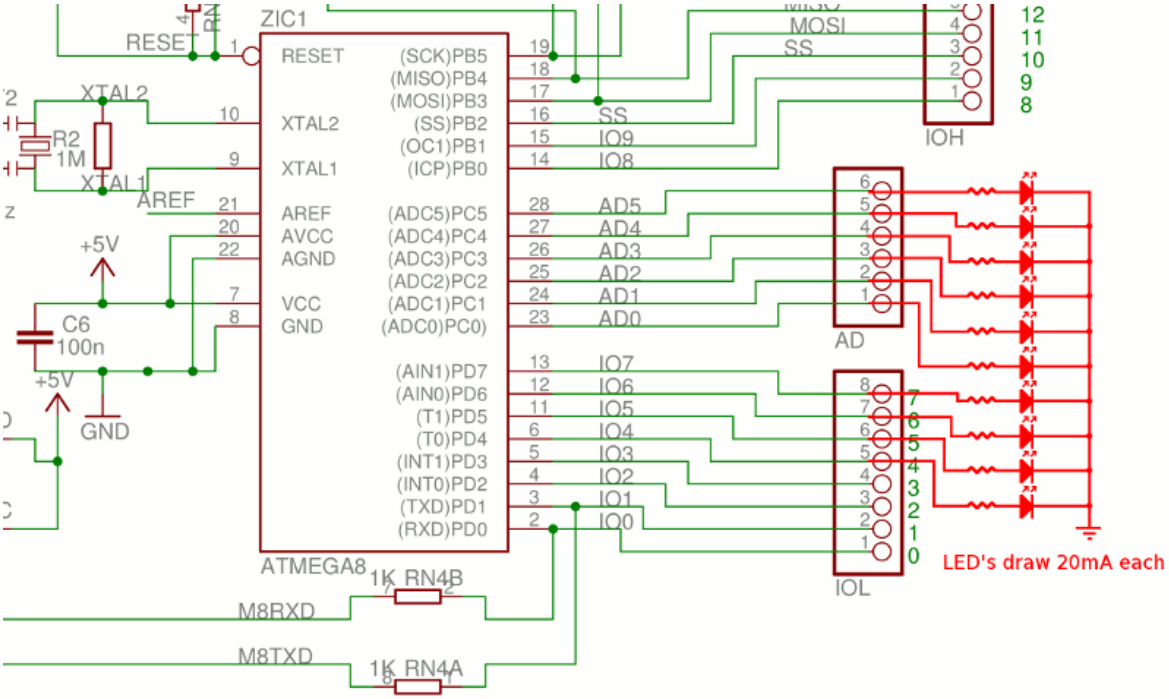
FORMA 10: SUPERAR LA TENSIÓN TOTAL QUE PUEDEN ENTREGAR LA SALIDAS.

¿CÓMO OCURRE?

Si el consumo de los pines es superior a 20mA (o la suma total de sus consumos supera los 200mA) se supera la carga máxima que es capaz de resistir el microcontrolador y este sufre daños.

¿POR QUÉ?

Se supera la tensión máxima que puede atravesar el microcontrolador y al no poder entregar más se daña.

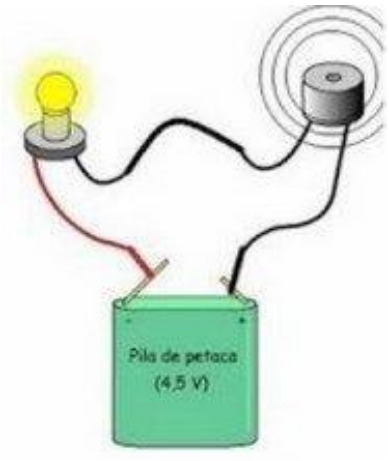


ANEXO 2: CIRCUITOS EN SERIE Y PARALELO

CIRCUITOS EN SERIE

LAS CARACTERÍSTICAS DE LOS CIRCUITOS EN SERIE SON:

Los elementos están conectados como los eslabones de una cadena (el final de uno con el principio del otro). La salida de uno a la entrada del siguiente y así sucesivamente hasta cerrar el circuito. Veamos una bombilla y un timbre conectados en serie:



Todos los elementos que se conectan en serie tienen la misma intensidad, o lo que es lo mismo, la misma intensidad recorre todos los elementos conectados en serie. Fíjate que la intensidad que sale de la pila es la misma que atraviesa cada receptor.

It = I1 = I2 = I3 .....

La tensión total de los elementos conectados en serie es la suma de cada una de las tensiones en cada elemento:

Vt = V1 + V2 + V3 ....

La resistencia total de todos los receptores conectados en serie en la suma de la resistencia de cada receptor.

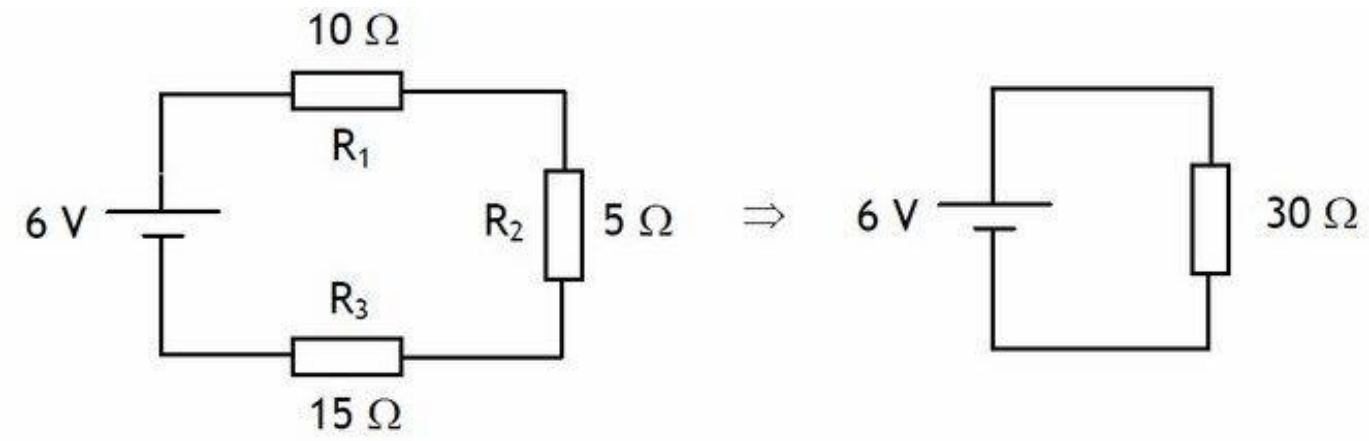
Rt = R1 + R2 + R3 .....

*Si un elemento de los conectados en serie deja de funcionar, los demás también. Date cuenta que si por un elemento no circula corriente, al estar en serie con el resto, por los demás tampoco ya que por todos pasa la misma corriente o intensidad (es como si se cortara el circuito).

Veamos como se resuelve un circuito en serie con 3 resistencias.

EJERCICIOS DE CIRCUITOS EN SERIE

Lo primero será calcular la resistencia total. Esta resistencia total también se llama **resistencia equivalente**, por que podemos sustituir todos las resistencia de los receptores en serie por una sola cuyo valor será el de la resistencia total. Fíjate en el circuito siguiente:



Rt = R1 + R2 + R3 = 10 + 5 + 15 = 30Ω. El circuito equivalente quedaría como el de la derecha con una sola resistencia de 30 ohmios. Ahora podríamos calcular la Intensidad total del circuito. Según lal ey de ohm:

It = Vt/Rt = 6/30 = 0,2 A //Que resulta que como todas las intensidades en serie son iguales:

It = I1 = I2 = I3 = 0,2 A //Todas valen 0,2 amperios.

Ahora solo nos queda aplicar la ley de ohm en cada receptor para calcular la tensión en cada uno de ellos:

V1 = I1 x R1 = 0,2 x 10 = 2V

V2 = I2 x R2 = 0,2 x 5 = 1V

V3 = I3 x R3 = 0,2 x 15 = 3V

Ahora podríamos comprobar si efectivamente las suma de las tensiones es igual a la tensión total:

Vt = V1 + V2 + V3 = 2 + 1 + 3 = 6 V //Como ves resulta que es cierto, la suma es igual a la tensión total de la pila 6 Voltios.

Recuerda: Para tener un circuito resuelto por completo es necesario que conozcas el valor de R, de I y de V del circuito total, y la de cada uno de los receptores. En este caso sería:

Vt, It y Rt

V1, I1 y R1

V2, I2 y R2

V3, I3 y R3

Como ves ya tenemos todos los datos del circuito, por lo tanto ¡Ya tenemos resuelto nuestro circuito en serie!.

Puede que nos pidan calcular las potencias en el circuito. En este caso sabiendo la fórmula la potencia que es:

P = V x I Pt = Vt x It = 6 x 0,2 = 1,2w

P1 = V1 x I1 = 2 x 0,2 = 0,4w

P2 = V2 x I2 =1 x 0,2 = 0,2w

P3 = V3 x I3 = 3 x 0,2 = 0,6w

Fíjate que en el caso de las potencias la suma de las potencias de cada receptor siempre es igual a la potencia total (en serie y en paralelo) **Pt = P1 + P2 + P3.**

Si no s piden la energía consumida en un tiempo determinado solo tendremos que aplicar la fórmula de la energía:

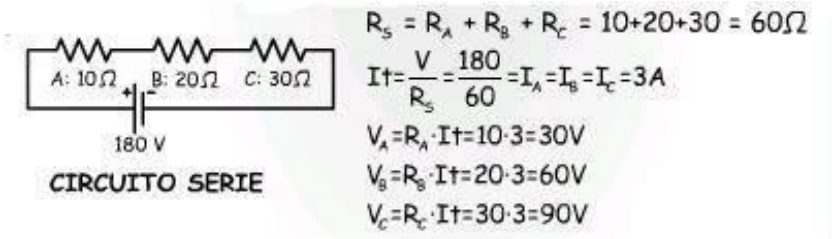
E = P x t. Por ejemplo vamos hacerlo para 2 horas.

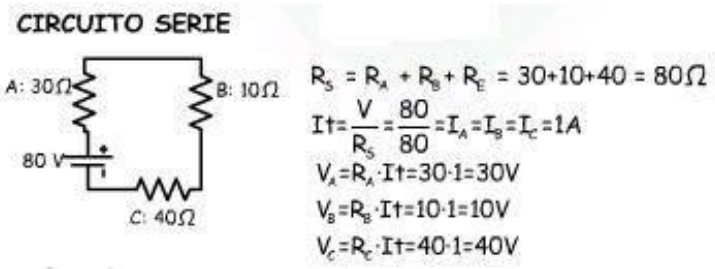
Et = Pt x t = 1,2 x 2 = 2,4 wh (vatios por hora). Si nos piden en Kwh (kilovatios por hora) antes de aplicar la fórmula tendremos que pasar los vatios de potencia a kilovatios dividiendo entre mil.

Pt = 0,0012 x 2 = 0,0024Kwh

También podríamos calcular las energía de cada receptor: **E1 = P1 x t ; E2 = P2 x t ....,** pero eso ya lo dejamos para que lo hagas tu solito.

Aquí tienes otros dos circuitos en serie resueltos:



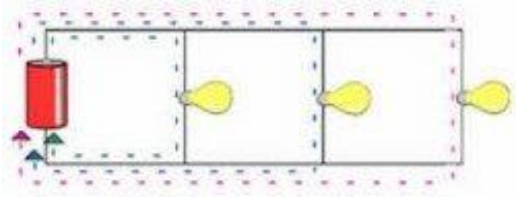


Ojo que no te despiste la colocación de las resistencias en el segundo circuito, si te fijas están una a continuación de otra, por lo tanto están en serie.

CIRCUITOS EN PARALELO

LAS CARACTERÍSTICAS DE LOS CIRCUITOS EN PARALELO SON:

Los elementos tienen conectadas sus entradas a un mismo punto del circuito y sus salidas a otro mismo punto del circuito.



Todos los elementos o receptores conectados en paralelo están a la misma tensión, por eso:

Vt = V1 = V2 = V3 .....

La suma de la intensidad que pasa por cada una de los receptores es la intensidad total:

It = I1 + I2 + I3 .....

OJO no te confundas, si te fijas es al revés que en serie.

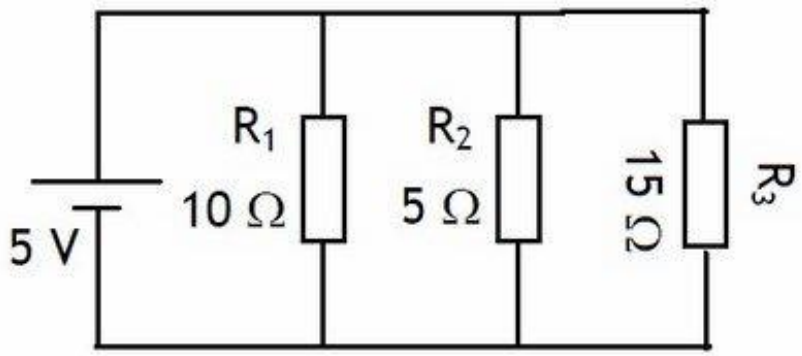
La resistencia total o equivalente de los receptores conectados en paralelo se calcula con la siguiente fórmula:

$$Rt = \frac{1}{\frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3} + \ldots}$$

Si un receptor deja de funcionar, los demás receptores siguen funcionando con normalidad. Este es el principal motivo por lo que la mayoría de los receptores se conectan en paralelo en las instalaciones.

Vamos a calcular un circuito en paralelo.

EJERCICIOS CIRCUITOS EN PARALELO



Podríamos seguir los mismos pasos que en serie, primero resistencia equivalente, luego la It, etc. En este caso vamos a seguir otros pasos y nos evitaremos tener que utilizar la fórmula de la resistencia total.

Sabemos que todas las tensiones son iguales, por lo que:

Vt = V1 = V2 = V3 = 5V; // todas valen 5 voltios.

Ahora calculamos la intensidad en cada receptor con la ley de ohm **I = V / R.**

I1 = V1 / R1 = 5/10 = 0,5ª

I2 = V2 / R2 = 5/5 = 1ª

I3 = V3 / R3 = 5/15 = 0,33ª

La intensidad total del circuito será la suma de todas las de los receptores.

It = I1 + I2 + I3 = 0,5 + 1 +0,33 = 1,83A

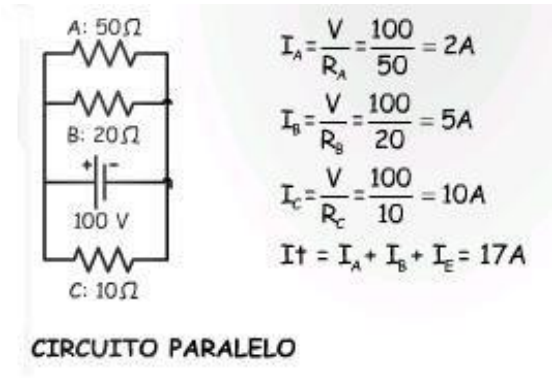
Date cuenta que la I3 realmente es 0,333333333… por lo que cometeremos un pequeño error sumando solo 0,33, pero es tan pequeño que no pasa nada.

¿Nos falta algo para acabar de resolver el circuito? Pues NO, ¡Ya tenemos nuestro circuito en paralelo resuelto! ¿Fácil no?

Repito que podríamos empezar por calcular Rt con la fórmula, pero es más rápido de esta forma. Si quieres puedes probar de la otra manera y verás que te dará lo mismo.

Para calcular las potencias y las energías se hace de la misma forma que en serie.

Aquí te dejamos otro circuito en paralelo resuelto:



ANEXO 3: DIVISOR DE TENSIÓN

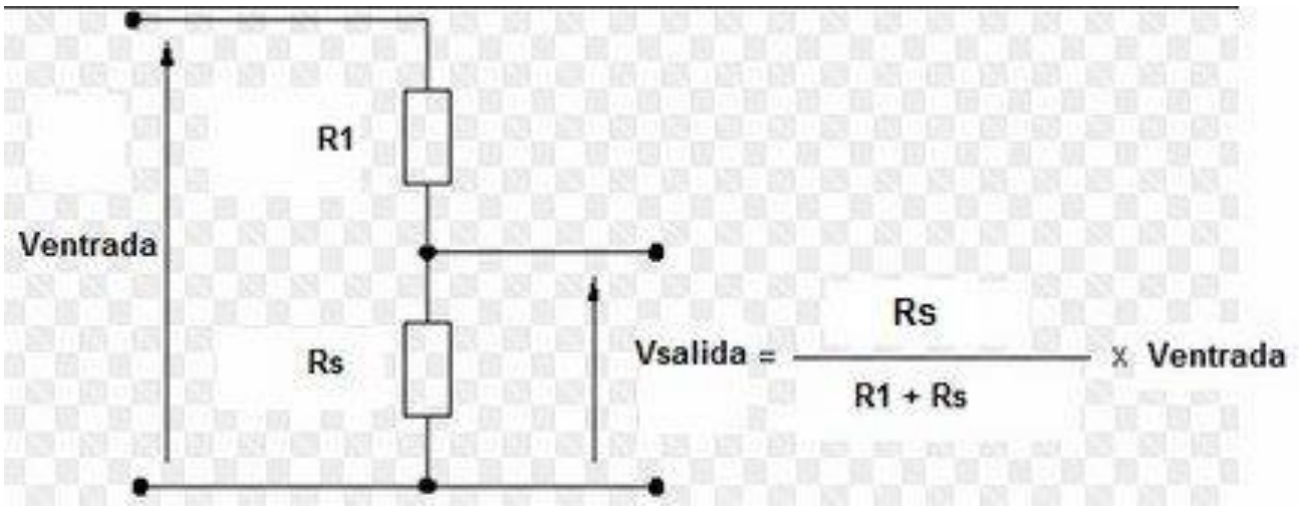
Es un circuito que divide la tensión de entrada en otras dos diferentes y más pequeñas.

En electrónica y electricidad se usa para alimentar (proporcionar tensión de alimentación) a un aparato, con una tensión más pequeña que la que proporcionan las pilas o baterías disponibles.

Imagina que queremos alimentar una bombilla a 6V y solo disponemos de una pila de 10V. ¿Qué haremos?. Pues la respuesta es muy sencilla: **Un divisor de tensión.**

CIRCUITO DIVISOR DE TENSIÓN

El circuito es muy sencillo, solo necesitamos 2 resistencias y conectarlas en serie como en el siguiente circuito:



Como ves en el circuito del divisor de tensión, al lado del circuito ya viene una fórmula. ¿De donde sale esa fórmula?

Muy sencillo, vamos a demostrarla.

Las dos resistencias están en serie y por lo tanto:

Rt = R1 + Rs

Las intensidades del circuito, al ser en serie serán todas iguales:

Ie o Itotal = I1 = Is

La suma de las tensiones será igual a la tensión total:

Ve = V1 + Vs

Según la ley de ohm V = I x R; por lo que:

V1 = I1 x R1

Vs = Is x Rs

Como todas las Intensidades son iguales las vamos a llamar I, con lo que nos queda:

V1 = I x R1

Vs = I x Rs

Si ponemos estos valores en la fórmula Ve = V1 + Vs; tenemos:

Ve = (I x R1) + (I x Rs) ;

Ve = I x R1 + I x R2; //sacando factor común la I tenemos:

Ve = I x (R1 + Rs); //Si despejamos de esta fórmula la I tenemos que:

I = Ve / (R1 + Rs). Fórmula importante.

Con todo esto ahora vamos a calcular la tensión de salida con la ley de ohm:

Vs = I x Rs; //recuerda que I = Is. Sustituimos en esta fórmula el valor de I por el calculado anteriormente en la formula en negrita y nos quedará:

$$Vs = \frac{Ve}{(R1 + Rs)} \times Rs$$

Si cambiamos la Ve y la Rs de lugar, la ecuación es la misma, pero más bonita, por que dejamos las R por una lado y la V por otro. En definitiva, nos quedará la fórmula que queríamos demostrar al principio:

$$Vs = \frac{Rs}{(R1 + Rs)} \times Ve$$

Fíjate que R1 + Rs es igual a la resistencia total del circuito en serie (Rt), por eso a veces verás la fórmula con el divisor en lugar de R1 + Rs como Rt.

Igual que hemos calculado Vs podríamos calcular V1 y nos quedaría:

$$V1 = \frac{R1}{(R1 + Rs)} \times Ve$$

Aquí tienes el desarrollo completo de forma simplificada:

$$\mathbf{Ve} = IR_1 + IR_S = I(R_1 + R_S)$$

$$I = \frac{V}{(R_1 + R_S)}$$

$$V_S = IR_S = \frac{V}{(R_1 + R_S)}(R_S) = V\frac{R_S}{R_1 + R_S}$$

¿Y ahora qué? Pues ahora para montar nuestro divisor de tensión, conocemos el valor de Vs y el de Ve. Ponemos una resistencia cualquiera, elegida por nosotros, con un valor cualquiera para Rs, y lo que tendremos que hacer es despejar el valor de la otra resistencia R1

Veamos nuestro caso: **Bombilla a 6V con una pila de 10V**

Ve = 10V; Vs = 6V; lógicamente si Ve = V1 + Vs; si despejamos V1, será V1 = 4V. Como ves siempre la tensión de la resistencia de arriba será la tensión entrada menos la de salida (circuito en serie).

Imaginemos que ponemos una Rs de 100Ω (un valor cualquiera, o el valor de una resistencia que tengamos). En este caso ya podremos despejar R1 de la fórmula de Vs. Vamos hacerlo.

Aplicando la fórmula principal de Vs, pero despejando el denominador (R1 + Rs) al otro lado de la ecuación, tendremos:

Vs x (R1 + Rs) = Rs x Ve ===> Vs x R1 + Vs x Rs = Rs x Ve ===>

Vs x R1 = (Rs x Ve) - (Vs x Rs) con lo que despejando R1 ===>

R1 = [(Rs x Ve) - (Vs x Rs)]/ Vs; si ponemos valores obtenemos el valor de R1. Para nuestro caso:

R1 = [(100 x 10) - (6 x 100)]/ 6

R1= 400/6 = 66,666Ω.

Pues para construir nuestro divisor de tensión lo haremos con una resistencia R1 de valor 66,66Ω y una resistencia de salida Rs = 100Ω.

Si no te gusta esta forma, hay otra más sencilla. Primero calculamos la I con Vs y luego con la ley de ohm en V1 despejamos la R1:

Vs = I x Rs ==> I = Vs/Rs = 6/100 = 0.06A, Como esta I es iguala a la que paso por R1 y la V1 = 4V;

V1 = I x R1 ==> R1 = V1/I = 4/0,06 =66,66Ω; como ves sale lo mismo. Usa la forma que más te guste para calcular R1.

¿Funcionará de verdad? Comprobemos si es cierto.

En nuestro circuito con los datos obtenidos tenemos que:

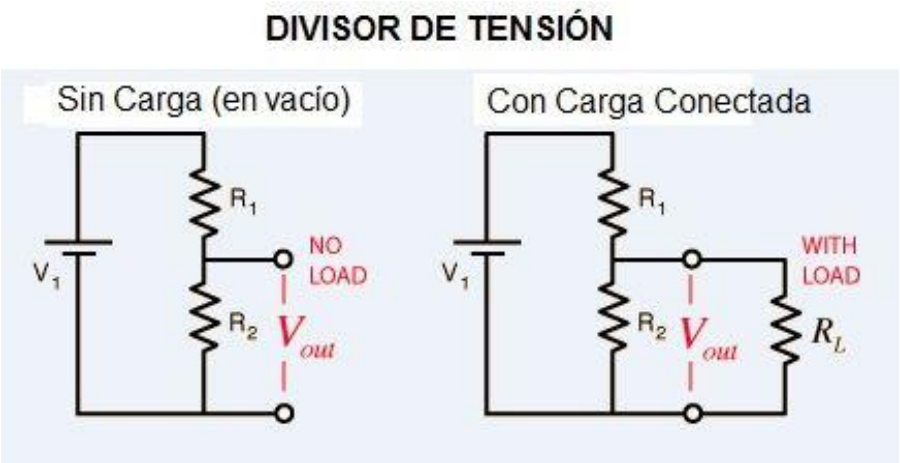
Ve = I x Rt; de donde I = Ve/Rt = 10/(66,66 + 100) = 0,06A (por el segundo método esto ya lo tendríamos calculado).

Con esa I; la Vs = I x Rs = 0,06 x 100 = 6V **¡¡¡Comprobado!!!** Tenemos a la salida los 6V que necesitamos.

Ahora cualquier cosa que conectemos en paralelo con Rs, precisamente por estar en paralelo con ella, quedará a la misma tensión a la que está Rs, es decir 6V. Ya podremos conectar nuestra bombilla a 6V, con una pila de 10V.

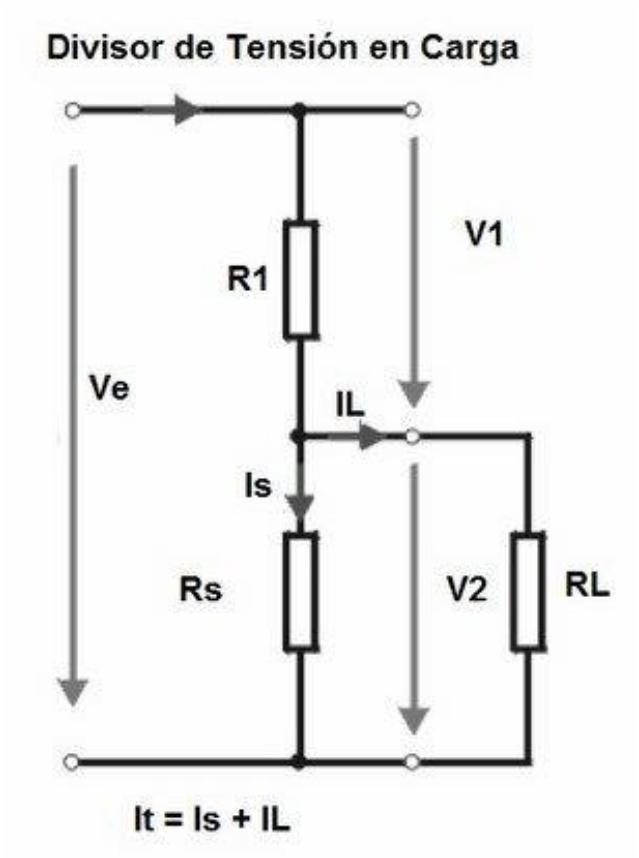
Como no existen resistencias de 66,66Ω pondremos la más cercana, por ejemplo de 60Ω con lo que tendremos un margen muy pequeño de error.

Todo esto está muy bien pero para trabajar en vacío, es decir sin carga. ¿Qué pasa cuando conectamos la carga (la bombilla) en el circuito. Pues que todo cambia.



Entonces.... ¿Problema resuelto? Pues depende. Para la mayoría de los casos esta forma teórica valdría, de hecho es la que más se usa, pero en la práctica real no.

Al conectar el receptor, este tendrá una resistencia (RL) y el problema es que esta resistencia cambiará la I total del circuito, ya no será la I teórica calculada anteriormente, y si cambia la intensidad total del circuito, las caídas de tensión en R1 (la V1) y en Rs (la Vs) también cambiarán, ya que dependen de la I, modificando los valores de Vs, que ya no serán los 6V que realmente necesitamos. Fíjate como quedaría el divisor al poner la carga (bombilla):



¿Qué hacemos si queremos nuestro divisor de tensión real? ¡¡¡Tranquilidad Hay una solución!!!

DIVISOR DE TENSIÓN EN CARGA

La solución es que calculemos primero la corriente que va a demandar nuestro receptor (la bombilla) y hacer los cálculos para el divisor en vacío, pero para una corriente 10 veces mayor.

La pila de 10V sería la entrada (Ventrada) y la bombilla funcionará a 6 V. La bombilla la colocaríamos en la tensión de salida (Vsalida). Lógicamente tenemos que conocer la intensidad que demandará nuestro aparato al conectarlo para que funcione correctamente. En nuestro caso la bombilla imaginemos que tiene una intensidad de 1mA (miliamperio) = 0,001A.

Datos iniciales: Ve, Vs, e Intensidad que consume el aparato de salida (en nuestro caso la bombilla)

A veces no nos dan el valor directamente de la intensidad, pero si no es así, nos darán la potencia del aparato. Imagina que nuestra bombilla es de 0,006w. ¿Cual será su intensidad?.

Sabiendo la fórmula de la potencia es muy fácil calcular la intensidad, ya que sabemos la tensión a la que estará conectada, en nuestro caso a 6V:

Potencia = V x I; despejando I = P/V. En nuestro caso I = 0,006w/6 = 0,001A = 1mA

Una vez que conocemos la intensidad que consume nuestro receptor, hacemos los cálculos del divisor del tensión en vació pero para una supuesta intensidad 10 veces mayor, es decir para una I de 0,001 x10 = 0,01A.

Ya tenemos los datos iniciales. Ve = 10V; Vs = 6V; I = 0,01A. Ahora veamos como construimos el divisor de tensión.

El problema es saber de que valor tendrían que ser las resistencias para que con 10V a la entrada obtengamos 6V a la salida con una intensidad de 0,01A. OJO estos 3 datos son siempre imprescindibles.

Es muy fácil, solo hay que aplicar la ley de ohm en cada resistencia. V1 = R1 x I; despejamos R1;

R1 = V1/I = 4/0,01 = 400Ω

de la misma forma Vs = Rs x I;

despejando Rs = Vs / I = 6/0,01 = 600Ω

Si montamos nuestro divisor con esas dos resistencias, problema solucionado.

Comprobemos que es cierto.

¿Qué tensión tendremos en Rs?

Apliquemos la ley de ohm: Vs = Rs x I = 600Ω x 0,01A = 6V. ¡¡¡Perfecto!!!

En V1 tendremos V1 = R1 x I = 400 x 0,01 = 4V

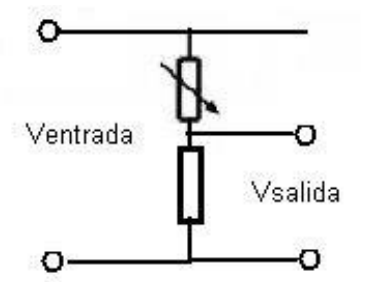
¿Qué pasará ahora cuando conectemos nuestro receptor (bombilla)?

Ahora aunque conectemos la bombilla a la salida, como el divisor está calculado para intensidades muy superiores, la intensidad que aumenta la total teórica del circuito en vacío es la de la bombilla 0,001A. Este aumento de intensidad no cambiará mucho los valores de V1 y Vs por que es muy pequeña (10 veces menor) respecto a la que circulaba en vacío, no produciendo grandes cambios.

Ahora sí ¡¡¡Problema solucionado!!!

DIVISOR DE TENSIÓN CON TENSIONES VARIABLES

Una forma sencilla de construir un divisor de tensión que nos valga para varias tensiones de salida diferentes es simplemente poner un potenciómetro en donde colocábamos la R1:



Fíjate que ahora solo cambiando el valor del potenciómetro cambiaremos la tensión de salida. Aumentando o disminuyendo la resistencia del potenciómetro, aumentaremos o disminuiremos la tensión de salida. Ya tenemos nuestro divisor de tensión variable o regulador de tensión.